Microsoft System Journal

ISSN 0933-9434 | Mai/Juni 1988 | 2. Jg./Heft 3

ÖS 150,- | DM 19,80 | sfr 19,80

Von MS-DOS nach Windows, von Windows zum MS-OS/2-Presentation Manager

Die Zukunft von MS-OS/2

Optimales Testen mit CodeView

Tools und Tricks für C-Programmierer

Liebe Leser,

MS-OS/2 wird ausgeliefert! Vorerst freilich erst die Version 1.0 ohne Presentation Manager. Aber auch der »PM« ist unterwegs. Im neuen Update des Software Development Kits ist eine Vorabversion für Entwickler schon enthalten. Auch die ersten MS-OS/2-Applikationen für die Version 1.0 sind schon auf dem Markt. Auf der Cebit im März konnte man ja schon einiges sehen. Die meisten Entwickler aber warten auf den Presentation Manager und schreiben für die Version 1.1 mit der standardisierten grafischen Bedieneroberfläche. Und viele Applikationen werden von vornherein für den Microsoft SQL Server und den MS-OS/2 LAN Manager angepaßt. Mit der Bereitstellung dieser neuen Netzwerktechnologien im Laufe das Jahres 1988 beginnt die heiße Phase der MS-OS/2-Softwareentwicklung. Diese heiße Phase wird von Microsoft durch drei verschiedene Großereignisse eingeläutet.

Im April stellt Microsoft den Systemprogrammierern die neuen Entwicklungswerkzeuge für MS-OS/2 zur Verfügung: die OS/2-Compiler C 5.1, Basic 6.0, Fortran 4.1, Pascal 4.0 und MASM 5.1, sowie den neuen Microsoft Editor und das MS-OS/2 Programmierer Toolkit.

Am 16. und 17. Mai veranstaltet Microsoft in Frankfurt das deutsche MS-OS/2 Symposium, auf dem Hard- und Softwarehersteller und unabhängige Systemhäuser ihre MS-OS/2-Strategie erläutern und die gemeinsame MS-OS/2 Zukunft diskutieren werden.

Vom 30. Mai bis zum 1. Juni findet dann in Düsseldorf die europäische MS-OS/2 LAN-Entwickler-Konferenz statt. Chefentwickler der Microsoft Corporation und der 3Com Corporation werden europäischen Netzwerk-Software-Entwicklern technische Einzelheiten zum Microsoft OS/2 LAN Manager erläutern. Auch die ersten OS/2 LAN-Applikationen aus den USA und Europa werden in einer Austellung zu sehen sein.

Das Microsoft System Journal wird über die neuen Compiler und Tools ebenso ausführlich berichten, wie über die beiden Kongresse. Unsere Redakteure werden sich in Frankfurt und Düsseldorf die interessanten Leute zu exklusiven Interviews suchen - damit die MSJ-Leser frühzeitig Einblicke in das nun schnell heraufziehende neue MS-OS/2-Zeitalter erhalten können.

Michael Kausch

Pressesprecher Microsoft GmbH

Ausgabe Mai/Juni 1988

Microsoft System Journal

Microsoft Windows

MS-OS/2

Microsoft C und QuickC

CodeView

COBOL

Rubriken

Als Vorbereitung auf den Presentation Manager:

Paradox unter Windows 2.0

Ansa Software, der kalifornische Softwarehersteller, der vor einiger Zeit von Borland International aufgekauft wurde, verkauft nur ein Produkt, doch dieses Produkt, die relationale Datenbank Paradox, wird in diesem Jahr in vier neuen Versionen erscheinen; eine fünfte wird wahrscheinlich Anfang 1989 erhältlich sein.

Die Sprößlinge zielen auf verschiedene Betriebssystemumgebungen: Microsoft Windows 2.0, OS/2 Presentation Manager, UNIX 5.3/XENIX und mit Hilfe von Utilities im geschützten Modus auf dem 80386 unter MS-DOS 3.x. Paradox 386 wurde bereits im vergangenen November auf der Comdex gezeigt und wird zum Erscheinungstermin dieser Ausgabe sicherlich verfügbar sein; diese Version benötigt keine besonderen Tools, nur ein 386-System. Die Version für OS/2 1.0 soll im ersten Quartal 1988 herausgebracht werden, die Windows-Version im zweiten Quartal, die UNIX/XENIX-Version im dritten Quartal und die Presentation-Manager-Version etwa Anfang 1989, was jedoch davon abhängt, wann der OS/2 Presentation Manager verfügbar sein wird.

Alle Paradox-Versionen, die alten wie die neuen, werden Multiuser-Fähigkeiten haben und datenkompatibel sein. Mit Versionen für alle bedeutenden PC-Umgebungen (bis auf den Macintosh - Ansa analysiert derzeit die Machbarkeit einer Mac-Version) ist Ansa in der Lage, die überwiegende Mehrzahl der Benutzer von Desktop-PCs zu erreichen. Und alle Benutzer an den verschiedenen Knoten eines Netzwerks werden auf eine gemeinsame Datenbank zugreifen und sie verändern können, unabhängig von der Hardware oder dem Betriebssystem, das sie verwenden.

Ein solider Entwurf zahlt sich aus

Was ist notwendig, um eine große, komplexe MS-DOS-Anwendung wie Paradox auf all diese verschiedenen Betriebssystemumgebungen zu portieren? In einem Interview mit dem Microsoft Systems Journal sagte Richard Schwartz, der Vizepräsident für Softwareentwicklung und einer der Gründer von Ansa, daß die Umsetzung von einer zeichenorientierten Umgebung auf eine andere zeichenorientierte ziemlich unkompliziert ist, was jedoch im wesentlichen auch der Voraussicht der Designer von Paradox zu verdanken ist.

»Wir haben Paradox von Anfang an so entworfen, daß es sehr sauber und portabel ist«, sagt Schwartz. »Wir haben nicht versucht, mit unsauberen Tricks auf das Betriebssystem zuzugreifen und haben auch anderweitig keine Hardwareabhängigkeiten entstehen lassen.« Das bedeutet natürlich nicht, daß das ursprüngliche Paradox nicht direkt in den Bildschirmspeicher schreibt. »Natürlich tun wir das, doch wir haben das isoliert. Wir haben ein Modul, das direkt auf den Bildschirmspeicher zugreift. Und auch die BIOS-Abhängigkeiten befinden sich alle an einer Stelle.«

Bild 1: *Die relationale Datenbank Paradox profitiert durch größere Flexibilität enorm von der Windows-2.0-Umgebung.*

Die Umsetzung von einer zeichenorientierten Umgebung (MS-DOS 3.x) in eine grafikorientierte Umgebung (Microsoft Windows und OS/2 Presentation Manager) ist ein erheblich ergeizigeres Unterfangen. Schwartz sprach über das Warum und Wie dieser Umsetzung und darüber, wie die grafikorientierte Version von Paradox aussehen, wie sie bedient und welche Leistung sie im Vergleich zum Original haben wird.

Warum Windows?

Die Umsetzung von Paradox auf Windows zum jetzigen Zeitpunkt wird Ansa, laut Schwartz, zahlreiche Vorteile beim Marketing und bei der Implementierung unter dem OS/2 Presentation Manager geben.

»Wir glauben, daß es sehr wichtig ist, eine Windows-Version zu haben; nicht weil wir Windows 2.0 als unseren Hauptmarkt ansehen - der Presentation Manager wird den Windows-Markt sicher verkleinern - sondern weil viele Firmen die Benutzerschnittstelle des Presentation Managers mit Windows 2.0 beurteilen werden. Und Anwendungen, die frühzeitig unter Windows laufen, werden einen großen Vorteil haben, wenn man die langen Beurteilungszyklen bedenkt, die in diesen Firmen durchlaufen werden.«

Zweitens ist Windows 2.0 jetzt verfügbar und der Presentation Manager nicht, so daß die Arbeit, Paradox für Windows zu überdenken und neu zu programmieren, für Ansa die beste Vorbereitung auf die spätere Portierung auf den Presentation Manager ist - auch wenn die Programmierschnittstelle des Presentation Managers sich erheblich von der von Windows unterscheiden wird.

Bilder 2 und 3: *Windows 2.0 ermöglicht es einem Paradox-Benutzer, mehrere Ansichten seiner Daten in verschiedenen Konfigurationen, je nach individuellen Anforderungen, zu positionieren.*

Gute Nachrichten

Wie schwierig war die Umstellung auf Windows? Laut Schwartz wurde alles bis auf die Basis, die Datenbankmaschine, neu entworfen, wobei einzelne Teile der vorherigen Architektur und Quellcode aus der Original-Version übernommen wurden. Trotzdem, so versichert er, wurden etwa 50 Prozent des Original-Quellcodes ohne Änderungen in diese neue Welt übernommen. Das ist ein noch während der Umsetzung geschätzter Wert, doch selbst wenn das zu optimistisch sein sollte, sind das doch gute Nachrichten für Windows-Entwickler: Nicht alle Programme müssen von Grund auf neu programmiert werden, um in der neuen Grafikumgebung laufen zu können.

Überdies wird die Umsetzung im Fall von Ansa wahrscheinlich etwa zwei Mannjahre erfordern. Das ist nicht schlecht, vor allem, wenn man berücksichtigt, daß das Original-Paradox insgesamt 14 Mannjahre Entwicklungsaufwand benötigte. Und die Firma hat dies erreicht, ohne Know-How dazukaufen zu müssen. Auch das sind gute Nachrichten, denn Macintosh- und Windows-Programmierer sind derzeit rar und haben einen sehr hohen Marktwert.

»Wir haben versucht, Leute einzustellen, die bereits mit Grafikumgebungen Erfahrungen haben. Wir haben dafür zeitweise einen Kopfjäger eingesetzt, doch wir waren nicht sehr erfolgreich«, sagt Schwartz dazu. »Und am Ende stellte sich heraus, daß wir sehr gut beraten waren, daß wir gute Softwareingenieure mit einem guten Informatik-Basiswissen nahmen und ihnen die Möglichkeit boten, sich dann unter Windows schnell weiterzuentwickeln. Wir glauben herausgefunden zu haben, das ein fundiertes Informatik-Basiswissen der wichtigste Faktor ist.«

Es paßt ins Konzept

Der Umstieg von Zeichen auf Pixel wurde Ansa dadurch erleichtert, daß Paradox trotz seiner weit zurückliegenden Ursprünge (die Entwicklung der ersten Version begann im Sommer 1981) immer schon ein sehr stark visuell orientiertes, interaktives Programm gewesen ist. Von der Benutzerschnittstelle her gesehen, ist die Umsetzung auf Windows also nur eine Erweiterung von Ideen, die bereits vorhanden waren, und keine grundlegende Umstrukturierung.

»Wir waren schon immer sehr stark visuell orientiert, oder besser gesagt, objektorientiert«, meint Schwartz. »Von daher paßte es einfach ins Konzept.«

Im zeichenorientierten Paradox werden Datentabellen dem Benutzer zum Beispiel in Form von visuellen Gebilden dargeboten, die sehr stark wie Fenster aussehen, nur daß die Ränder aus Grafikzeichen bestehen und nicht ganz herum gehen (*Bild 2*). Wie ein Frame in Framework oder ein Bildschirm voller Kalkulationstabellen sind diese Objekte Ausschnitte von größeren Datenbeständen; sie werden in Paradox Images genannt.

Paradox präsentiert dem Benutzer einen Arbeitsbereich, auf dem er mehrere Images gleichzeitig halten kann, fast genauso wie Windows das macht. Auch wenn eine Maus nicht unterstützt wird, erlaubt das zeichenorientierte Produkt dem Benutzer die Veränderung der Größe und der Position der Images und die Breite der Spalten darin, indem eine bestimmte Stelle ausgewählt und dann die Cursortasten verwendet werden. Diese Möglichkeit, einen Datenbereich sofort direkt zu bearbeiten, ohne dazu einen Menübefehl auswählen zu müssen, nennt Schwartz »direkte Bedienbarkeit«, ein wichtiger Bestandteil der Benutzerschnittstelle des Original-Paradox.

Bilder 4 und 5: *Die Dialogboxen und Buttons von Windows ermöglichen Paradox einen wesentlich verbesserten Kontakt zum Benutzer.*

Visuelle Interaktion

»Die Idee dabei war, so weit wie möglich und überall wo machbar die visuelle Interaktion anstelle der beschreibenden Interaktion zuzulassen«, erläutert Schwartz. »Anstatt vom Benutzer zu verlangen, daß er etwas erläutern oder beschreiben soll, fordern wir ihn nur auf, etwas zu zeigen. Das paßt sehr gut in die Windows-Umgebung.«

Paradox unter Windows wird auf den ersten Blick der Original-Version ziemlich ähnlich sehen (*Bild 3*), nur daß die Lotus-artigen Menüs durch Drop-Down-Menüs ersetzt wurden, der Text schwarz auf weiß dargestellt wird und die Images komplette Dokumentfenster sind, mit Titelzeile, Scrollbalken, Steuermenüs und Größensinnbild. Die direkte Bedienbarkeit ist noch immer vorhanden, sie ist sogar stärker vertreten. Natürlich wird auch die Maus unterstützt.

Ein neues Beispiel für die direkte Bedienbarkeit von Windows-Paradox ist die Angabe von Sortierungen. In der zeichenorientierten Version gab der Benutzer das Feld, das Sortierschlüssel sein sollte und die Sortierreihenfolge dadurch an, daß er die Feldnummer gefolgt von einem »A« (ascending, aufsteigend) oder einem »D« (descending, absteigend) eingab (*Bild 4*).

Die Windows-Version (*Bild 5*) vereinfacht diesen Vorgang und macht ihn weniger analytisch, indem der Benutzer jedes Schlüsselfeld für die Sortierung direkt mit einem Doppelklick auswählen kann. Ein Pfeil, der die Sortierfolge angibt (auf- oder absteigend) erscheint dann links neben dem gewählten Feldnamen; um die Sortierreihenfolge zu ändern, braucht der Benutzer nur die Richtung des Pfeils zu ändern indem er ihn mit Doppelklick umschaltet.

Diese Art, auf einen Wert zu zeigen und ihn umzuschalten oder aus Menüs auszuwählen gibt es auch an anderen Stellen im Windows-Paradox. Ein Beispiel: Bei der Anlage oder Umstrukturierung einer Tabelle, wo der Benutzer nach einem Datentyp gefragt wird, kann er mit der Maus klicken und sich ein Menü mit den verfügbaren Datentypen anzeigen lassen.

Ähnlich kann sich der Benutzer bei diesem Vorgang mit einem Doppelklick auf einen Feldnamen eine Dialogbox anzeigen lassen, in der die momentanen Definitionen für Gültigkeitsprüfungen angezeigt oder neue angelegt werden können.

Vorteile dieser Implementation

Bei seiner Beschreibung des Umstellungsaufwands spielt Schwartz die gefürchtete Windows-Lernkurve herunter und betont statt dessen die Vorteile, die durch die grafische Oberfläche zustandekommen. Im besonderen verweist er auf die größere Unabhängigkeit der verschiedenen Objekte mit denen der Anwender umgeht.

In der Windows-Umgebung existiert jede Tabelle im Arbeitsbereich (ebenso wie jede Masken-, Abfrage- und Listendefinition) in einem eigenen Fenster mit eigener Meldungsroutine. Diese Meldungsbehandlungsroutine ist dafür verantwortlich, alle sie betreffenden Vorgänge zu kennen, z.B. wie das jeweilige Fenster neu angezeigt und aktualisiert wird. In der zeichenorientierten Version mußte stärker darauf geachtet werden, in welchem »globalen Zustand« sich das Programm gerade befand, das heißt, wie Änderungen einer Tabelle alle anderen Dinge im Arbeitsbereich beeinflussen.

»Auf der Implementierungsebene bedeutet das,« meint Schwartz, »daß das Programm so aufgebaut ist, das alles strikt lokal zum einzelnen Image ist. So kann man eine lokale Routine haben, die weiß, wie die Fenstergröße geändert wird oder welche Bereiche des Fensters neu gezeichnet werden müssen. Wenn irgendetwas im Bereich dieses Fensters verändert wird, achtet Windows darauf, daß Meldungen an alle Fenster geschickt werden, damit sie wissen, daß ein Bereich neu angezeigt werden muß.«

Schwierigkeiten beim Debuggen

Laut Schwartz war es eine ziemlich starke Behinderung der Entwicklung der Windows-Version von Paradox, daß Debug-Tools für Windows noch sehr rar sind.

»Es handelt sich dabei sowieso um ein größeres Problem,« sagt Schwartz dazu, »denn in der Windows-Umgebung spielt sich sehr viel hinter den Kulissen ab.« Die Verschiebbarkeit (moveable) und die Aufgebbarkeit (discardable) von Windows-Objektcode verkompliziert die Dinge und führt dazu, daß Bugs schwerer zu finden sind. Die Bugs treten erst sehr viel später im Entwicklungsprozeß als Ergebnis der Systembelastung auf, zum Beispiel weil ein Programmteil im Speicher verschoben wurde.

Die derzeitige Version von CodeView läuft nicht in der Windows-Umgebung, Microsoft arbeitet jedoch an einer Windows-Version. SymDeb wird mit dem Windows Softwareentwicklungskit geliefert, doch Ansa hat ihn nicht besonders häufig verwendet; sie haben sich mehr auf das Debuggen auf der Ebene der Algorithmen verlassen, das heißt ganz einfach printf-Anweisungen verwendet, um Variablenwerte an verschiedenen Stellen im Programm zu testen.

»In unserer Entwicklungsumgebung haben wir an jedes System sowohl einen monochromen als auch einen EGA-Bildschirm angeschlossen. Wir haben es so arrangiert, daß wir printf-Anweisungen in das Programm einfügen können, die dann auf dem monochromen Monitor ausgegeben werden. So wird der Effekt des traditionellen Debuggens in die Windows-Umgebung eingebracht.«

Das Debuggen wurde dadurch noch schwieriger gemacht, daß Ansa mit frühen Vorabversionen von Windows und dem Windows-Entwicklungstools arbeitete (das Projekt begann im April 1987). Die Programmierer hatten oftmals große Schwierigkeiten herauszufinden, an welchen Problemen ihr Code und an welchen der Compiler oder die Umgebung schuld war.

Anwendervorteile

Die Unabhängigkeit der Bildschirmobjekte bringt Anwendern viele Vorteile. Am wichtigsten ist wohl, daß der Benutzer mehrere Dinge gleichzeitig durchführen kann und trotzdem noch den Überblick behält. Es gibt keine Einschränkungen mehr in der Anzahl der Images, die hintereinander auf den Bildschirm passen; der Anwender kann den Bildschirm mit überlappenden Fenstern füllen und dadurch die Informationsdichte erreichen, die er am besten findet.

Dazu gehören auch solche Vorgänge wie der Wechsel von einer Tabelle zur anderen, das Kopieren von Daten und Strukturen von einer Tabelle zur anderen, das Erkennen von Zusammenhängen zwischen einem Datenbestand und einem anderen und so weiter. Das bedeutet auch, daß Windows-Paradox für den Anwender weniger modusorientiert aussieht als der zeichenorientierte Vorgänger. Es gibt im Original-Paradox Fälle, in denen der Anwender (wenn auch nur kurz) einen Kontext verlassen muß, um in einem anderen zu arbeiten. Die Windows-Version verringert solche Moduswechsel und ihre Auswirkungen auf den Arbeitsablauf.

Ein Beispiel dafür: Wenn der Benutzer in der zeichenorientierten Version eine Dateneingabemaske bei der Dateneingabe nebenbei verändern möchte, gibt er einen Befehl ein, um die Maskenroutinen aufzurufen. Die Daten und die Maske, mit denen er gerade gearbeitet hat, werden erst wieder sichtbar, wenn er die Änderung der Bildschirmmaske beendet hat. In der Windows-Version erscheint die Maskendefinition in einem Unterfenster, wobei die Daten im übergeordneten Fenster sichtbar bleiben.

Zu den weiteren Vorteilen, die Windows-Paradox dem Anwender bietet, gehören:

- Die Windows-Version unterstützt automatisch alle hochauflösenden Ganzseitenbildschirme, die auch von Windows unterstützt werden.
- Die Listenroutinen bieten erweiterte Font-Unterstützung und akzeptieren auch Bitmap-Bilder aus der Zwischenablage (clipboard) von Windows. Der Anwender kann Paradox-Listen mit Logos, Unterschriften, Bildern und dergleichen verschönern. Windows-Paradox akzeptiert Bitmap-Bilder nicht als Datentypen, doch Schwartz gibt zu, daß das »eine natürliche Richtung« ist, in die sich das Produkt entwickeln wird.
- Windows-Paradox kann mit Hilfe von DDE (dynamic data exchange) Daten mit anderen Windows-Anwendungen austauschen. Bei der Vorstellung von Microsoft Excel in den USA zeigte Ansa eine frühe Version von Paradox, die als Auftragnehmer (client) unter Microsoft Excel lief, wobei Paradox mit der Excel-Makrosprache gestartet wurde und Daten über DDE an Microsoft Excel übergeben wurden (*Bild 6*). Die fertige Windows-Version von Paradox ermöglicht auch eine Umkehrung dieses Vorgangs: Man kann andere Anwendungen mit Programmen, die in der Programmiersprache von Paradox (PAL: Paradox application language) geschrieben wurden, starten und steuern.

Performance

Das ganze sieht recht rosig aus, doch Schwartz räumt ein, daß die Vorteile der Windows-Umgebung gleichzeitig zu einer Leistungsverschlechterung führen. Der Grund dafür liegt weniger darin, daß Windows Text im Grafikmodus darstellt, sondern vielmehr darin, daß Windows Programm-Module aus dem Speicher auslagert und sie später wieder einliest.

Da das Programm zum Zeitpunkt des Interviews noch in der Entwicklung war, konnte Ansa noch keine genauen Benchmark-Ergebnisse vorlegen. Schwartz meint jedoch,

Bild 6: *Paradox kann mit Hilfe von DDE Echtzeit-Kommunikationen mit anderen Anwendungen aufbauen.*

daß die Firma »mit der Performance auf einem 10-MHz-286-System und besseren Systemen sehr zufrieden ist«. Und er merkt dazu an, daß der SMARTDrive-Treiber, eine Disk-Cache-Utility von Microsoft zur Performance-Verbesserung von Windows, die Geschwindigkeit auf allen Systemen erheblich verbessert.

Die Furcht genommen

Was kann Richard Schwartz anderen Windows-Entwicklern für Ratschläge mit auf den Weg geben? Am wichtigsten ist wohl, das der Neuentwurf eines MS-DOS-Programms für die Windows-Umgebung nicht die schreckliche Aufgabe ist, für die es allgemein gehalten wird, besonders dann nicht, wenn das Programm gut strukturiert ist. Das zweite besteht darin, wie man Individualität unter Bedingungen wahren kann, die eigentlich Gleichheit verlangen.

»Einige Leute meinen, daß alle Produkte gleich aussehen werden, daß dem Programmdesigner in der Windows-Umgebung alle Kreativität genommen worden ist«, sagt Schwart dazu. »Es ist so viel standardisiert worden, das alles gleich aussehen und gleich funktionieren wird. Das ist in gewisser Hinsicht richtig. Doch es gibt immer noch ganz unterschiedliche Arten, auf die man diese Umgebung zur Präsentation einer bestimmten Anwendung vorteilhaft einsetzen kann.«

»Sie müssen sich zunächst mit den Grundlagen beschäftigen, wie eine stark benutzerorientierte Anwendung aufgebaut wird und wie Sie sie präsentieren wollen. Und erst danach sehen Sie sich an, was es dafür an Windows-Möglichkeiten gibt und wie man sie einsetzt. Die größten Auswirkungen auf den Benutzer hat das darunterliegende Programm-Modell, und das wird von Windows nicht berührt. Windows ist nur ein Weg es darzustellen und es dem Anwender effektiver zu präsentieren.«

Craig Stinson/jü

Impressum

Das *Microsoft System Journal* erscheint alle zwei Monate (ungerade Monatszahlen) etwa Mitte des Vormonats.

Herausgeber, verantwortlich und Anschrift der Redaktion:
Microsoft GmbH, Redaktion *Microsoft System Journal*,
Erdinger Landstr. 2, D-8011 Aschheim-Dornach
Telefon: 089 / 46107-0, Teletex/Telex: (17) 89 83 28, Telefax: 90 63 55
Redaktion:
Günter Jürgensmeier, Haar (jü), Hartmut Niemeier, Wildenberg (ni)
Mitarbeiter dieser Ausgabe:
Kaare Christian, Richard Friedrich, Michael Geary, Allen Holub, Günter Jürgensmeier, Michael Kausch, Hartmut Niemeier, D. Norris, M.J. O'Leary, Charles Petzold, Joachim Schneider, Craig Stinson
Manuskripteinsendungen:
Manuskripte und Programmlistings werden von der Redaktion gerne angenommen. Mit der Einsendung von Manuskripten und Listings gibt der Verfasser die Zustimmung zum Abdruck und zur Vervielfältigung der Programmlistings auf Datenträgern. Honorare nach Vereinbarung. Für unverlangt eingesandte Manuskripte und Listings wird keine Haftung übernommen. Nicht zur Veröffentlichung gelangte Manuskripte und Listings können nur zurückgeschickt werden, wenn Rückporto beiliegt.
Titelgestaltung: Hermann Menig
Anzeigenverkauf: Marianne Nuß
Druck und Abonnements:
schury praxisformulare GmbH, Postfach 270, D-8200 Rosenheim
Bezugspreise: Das Einzelheft kostet DM 19,80. Der Abonnementpreis beträgt DM 115,- für 6 Ausgaben und DM 210,- für 12 Ausgaben. Zu den einzelnen Ausgaben ist zum Preis von DM 19,80 eine Diskette mit allen Listings erhältlich. Das Abonnement inklusive Diskette kostet DM 230,- für 6 Ausgaben und DM 420,- für 12 Ausgaben. In den Preisen enthalten sind Mehrwertsteuer, Versandkosten und Zustellgebühren. Auslandsbezug auf Anfrage. Sollte die Zeitschrift aus Gründen, die nicht vom Herausgeber zu vertreten sind, nicht geliefert werden können, besteht kein Anspruch auf Nachlieferung oder Erstattung vorausbezahlter Bezugsgelder.
Bezugsmöglichkeiten: In Buchhandlungen und im Computer-Fachhandel. Abonnements und Einzelbestellungen: schury GmbH.

Das *Microsoft System Journal* wird mit Microsoft Word 4.0 geschrieben und gestaltet. Der Ausdruck erfolgt mit den HP-Softfonts AD und AF und dem Programm- und Schriftenpaket *DocuJet* auf einem HP-LaserJet Series II.

Paradox unter MS-OS/2

Paradox wurde in Zeiten begrenzter Ressourcen konzipiert. Die Entwicklung begann in der ersten Hälfte des Jahres 1981, Monate bevor der IBM-PC angekündigt wurde und als 64 Kbyte noch ein Wunschtraum der meisten Computerbenutzer war.

In den nächsten Jahren, in denen Maschinen mit größeren Kapazitäten allgemein zugänglich wurden, wuchs Paradox, sowohl was das Anwendungsgebiet als auch was die Fähigkeiten anbelangt; als Paradox dann im Herbst 1985 angekündigt wurde, benötigte es 512 Kbyte, was in der Praxis bedeutete, daß ein Personalcomputer mit 640 Kbyte benötigt wird.

Doch wie jemand, der auch schon schlechtere Zeiten gesehen hat, hat Paradox niemals vergessen, was es bedeutet, in »Hungerzeiten« zurechtzukommen. Von Anfang an beinhaltet das Produkt eine virtuelle Speicherverwaltung, die Daten auf der Platte auslagert, wenn das nötig ist. Eines der wichtigsten Verkaufsargumente ist jedoch die heuristische Vorgehensweise bei relationalen Abfragen. Ein wichtiger Aspekt dieses Prozesses ist das »Nachdenken« der Maschine darüber, welche Daten derzeit im Speicher und welche gerade ausgelagert sind.

Der große Adreßraum, den die 80286- und 80386-Chips im geschützten Modus bieten, bedeuten für Paradox also eine Art »Befreiung«, die durch vorläufige Leistungsstatistiken eindrucksvoll bestätigt werden (*Bilder 7 und 8*).

Nach den Zahlen zu urteilen, erledigt Paradox große Sortierungen und komplexe Abfragen sehr viel schneller als die MS-DOS 3.3-Version. Die 80386-Version erledigt Abfragen ungefähr genauso effizient, wie die OS/2-Version und sie ist beim Sortieren sogar etwas schneller.

Die vielleicht überraschendste Information dieser Zahlen ist, daß Paradox 2.0 im Real-Modus unter OS/2 (in der Kompatibilitätsbox von OS/2) schneller ist, als das gleiche Produkt auf der gleichen Hardware unter MS-DOS 3.3. Laut Microsoft laufen die meisten MS-DOS-Programme in der Kompatibilitätsbox etwa fünf Prozent langsamer. Richard Schwartz von Ansa führt die Leistungssteigerung auf die Tatsache zurück, daß Paradox sehr viele wahlfreie Dateizugriffe durchführt und dabei erheblich vom Disk-Caching von OS/2 profitieren kann.

Leistungsdaten für die Microsoft Windows-Version von Paradox waren bei der Erstellung dieses Artikels leider noch nicht verfügbar.

Sekunden	Abfrage	Sort
Paradox 2.0 unter DOS3.3	73	93
Paradox 2.0 OS/2-KompBox	59	71
Paradox unter OS/2	46	60
Paradox 386 unter DOS3.3	48	44

Bild 7: *Vorläufige Angaben über die Performance von drei neuen Paradox-Versionen, gemessen auf einem IBM PS/2 Model 80. Die Abfrage besteht darin, eine Tabelle mit 5000 Sätzen mit einer Tabelle von 10000 Datensätzen zu verbinden; der Sort verwendet eine Tabelle mit 500 Sätzen. Die Werte für Paradox OS/2 und Paradox 386 können unterschiedlich sein, wenn die endgültigen Produkte ausgeliefert werden.*

Leistungsverbesserung	Abfrage	Sort
Paradox 2.0 OS/2 KompBox	19%	24%
Paradox unter OS/2	37%	35%
Paradox 386 unter DOS3.3	34%	53%

Bild 8: *Die Leistungssteigerung der drei neuen Paradox-Versionen im Vergleich zu Paradox 2.0 unter MS-DOS 3.3. Diese Werte sind vorläufig und geben nicht unbedingt die wirkliche Leistung der später ausgelieferten Produkte wider.*

Die Umsetzung von Windows-Anwendungen auf den OS/2 Presentation Manager

Von Windows zum OS/2 Presentation Manager

Der Microsoft OS/2 Presentation Manager stellt die Vereinigung von zwei Technologien dar: der Betriebssystemumgebung Windows und OS/2, dem Betriebssystem für den geschützten Modus. Wie bei den meisten Hochzeiten gibt es auch beim Presentation Manager etwas Altes, etwas Neues, etwas Traditionelles und etwas Zukunftsweisendes.

Das *Alte* ist Microsoft Windows. In gewissem Sinne *ist* der Presentation Manager sogar Windows. Auch wenn es eine Reihe von Änderungen gegenüber der MS-DOS-Version von Windows gibt, sind fast alle Programmierkonzepte direkt in die neue Umgebung übernommen worden: meldungsgesteuerte Programmierung, Fensterfunktionen, Ressourcen usw. Wenn Sie wissen, wie Windows programmiert wird, ist der Presentation Manager ein Klacks.

Das *Neue* sind die geschützte Hardwareumgebung und OS/2. Es wäre sicherlich nicht unberechtigt zu sagen, daß dies die Umgebung ist, für die Windows eigentlich gedacht ist. OS/2 bietet echtes, preemptives Multitasking im Gegensatz zum einfachen simulierten Multitasking von Windows. Anwendungen brauchen nicht länger aufeinander zu warten, um die Steuerung zu erhalten. Innerhalb einer Anwendung kann die OS/2-Möglichkeit mehrerer gleichzeitig ausführbarer Threads sehr sinnvoll eingesetzt werden, wobei beispielsweise jedem Fenster ein eigener Thread zugeordnet werden kann. Die geschützte Hardwareumgebung erweitert die Speicheradressierungsfähigkeiten enorm und macht dadurch einige der bisher softwaremäßig ausgeführten Speicherverwaltungsaufgaben des MS-DOS-Windows überflüssig.

Das *Traditionelle* ist die Grafikprogrammierschnittstelle GPI (Graphics Programming Interface). Sie wurde von den Großrechnerstandards GDDM (Graphics Data Display Manager) und GCP (Graphics Control Program) übernommen. GPI ersetzt das Graphics Device Interface (GDI) von Windows durch einen umfangreicheren und flexibleren Satz von Grafikfunktionen.

Das *Zukunftsweisende* ist, wie sie vielleicht schon vermutet haben, IBM. Der Presentation Manager spielt eine wichtige Rolle als Bedienungsoberfläche der System-Anwendungs-Architektur (SAA) von IBM, zumindest für Systeme mit Grafikfähigkeiten. IBM hat ehrgeizige Pläne für SAA: eine einheitliche Benutzeroberfläche und Programmierschnittstelle auf einer breiten Palette von Systemen, von PCs bis zu Großrechnern. Irgendwann einmal wird es möglich sein, eine Presentation-Manager-Anwendung auf jedem System neu zu kompilieren und laufenzulassen. Für Systeme ohne Grafik wird eine Untermenge der Funktionen des Presentation Managers verfügbar gemacht, die eine ähnliche Benutzerschnittstelle bietet, jedoch zeichenorientiert arbeitet.

Bild 1: *Der Presentation Manager verwendet die gleiche Benutzerschnittstelle wie Windows 2.0.*

Das Programm SayWhat

Um die Unterschiede und Gemeinsamkeiten zwischen einer Presentation-Manager-Anwendung und einer Windows-Anwendung zu untersuchen, werde ich hier ein einfaches Beispielprogramm mit dem Namen SayWhat vorstellen. (Es handelt sich dabei um meinen Beitrag im Wettbewerb um die »dümmste Windows-Anwendung«.) Die Grundlage dafür war das Beispiel »Hello« im Windows Software Development Kit, das ich ein wenig »aufgebohrt« habe. Anstatt die Meldung »Hello« nur an einer Stelle anzuzeigen, wandert der Text zufällig im Fenster herum und ändert dabei seine Farbe. Wenn SayWhat zum Sinnbild gemacht wird, wird die Meldung Zeichen für Zeichen angezeigt, wobei das Zeichen im Sinnbild herumwandert. Mit einem Steuerungsfenster in Form einer moduslosen Dialogbox kann man den angezeigten Text ändern und es kann eingestellt werden, wie schnell er sich bewegt und wie oft er eine zufällig gewählte neue Richtung auswählt.

SayWhat demonstriert mehrere Programmiertechniken, die für Windows und den Presentation Manager sehr nützlich sind. Er zeigt auch einen Fehler dabei auf, wie viele Windows-Anwendungen den Bildschirm beschreiben und wie dies verbessert werden kann. Die Versionen für Windows und den Presentation Manager sind nebeneinander abgedruckt, damit sie besser verglichen werden können. Die Listings sind mit der Listingnummer und »w« für »Windows« und »pm« für »Presentation Manager« beschriftet.

Beim Vergleich der Listings ist deutlich zu sehen, daß die Grobstruktur beider Programme identisch ist. Die wichtigsten Aspekte von Windows sind auch beim Presentation Manager vorhanden: Meldungen, Fensterfunktionen, Anzeige (painting) usw. Im Detail sind jedoch zahlreiche Unterschiede vorhanden.

Bild 2: *SayWhat in Aktion.*

Der Einstieg

Als erstes haben beide Programme eine Haupt-Funktion, im Presentation Manager wird sie jedoch auch `main( )` genannt und nicht `WinMain( )`, wie bei Windows-Anwendungen. Das ist so, weil eine Presentation-Manager-Anwendung zunächst einmal eine ganz normale OS/2-Anwendung ist; was sie von anderen OS/2-Anwendungen unterscheidet, ist die Verwendung der Programmierschnittstelle (API, Application Programming Interface) des Presentation Managers.

Die Parameter, die an `main` übergeben werden, unterscheiden sich von denen, die in der Windows-Version an `WinMain` übergeben werden. Der Parameter `lpszCmdLine` wird durch die gewohnteren und praktischeren Parameter `argc` und `argv` ersetzt. Es gibt keinen Parameter wie `nCmdShow`; wenn das Hauptfenster einer Anwendung als normales sichtbares Fenster mit `WinCreateStdWindow` angelegt wird, werden alle notwendigen Dinge automatisch berücksichtigt, inklusive der Fenstergröße. Wenn Sie eine direktere Kontrolle der Fenstergröße benötigen, legen Sie einfach das Fenster unsichtbar an und stellen dann selbst die Größe und Position ein. Schließlich fehlen noch die Parameter `hPrevInstance` und `hInstance`: `hPrevInstance` wird überhaupt nicht verwendet und es gibt im Presentation Manager keine Entsprechung für die Funktion `GetInstanceData`. Unter OS/2 gibt es Vorkommen nicht im gleichen Sinn wie unter Windows. Unter OS/2 gibt es nur Prozesse und jeder Prozeß initialisiert seine eigenen Daten. Nachfolgende Vorkommen einer Anwendung haben genau die gleiche Initialisierung wie das erste Vorkommen.

Die Meldungsschleife

Die Haupt-Meldungsschleife eines Presentation-Manager-Programms ist nahezu identisch mit der Hauptschleife eines Windows-Programms. Bis auf einige Namensänderungen und die Verwendung des Parameters `hAB` besteht der einzige Unterschied darin, daß die Funktion `TranslateMessage` nicht verwendet wird. Da diese Funktion sowieso immer benötigt wurde, ist sie mit `WinGetMsg` zusammengelegt worden. Die Tastaturmeldungen des Presentation Managers sind sogar erheblich vereinfacht worden. Die Meldungen `WM_KEYDOWN`, `WM_KEYUP`, `WM_SYSKEYDOWN` und `WM_SYSKEYUP` sind alle nicht mehr vorhanden und die Informationen, die von Ihnen geliefert wurden, sind nun alle über `WM_CHAR` zu erhalten. Die Meldung `WM_CHAR` übergibt nun die unbearbeiteten Tastaturinformationen (raw), die sonst von der Meldung `WM_KEYDOWN/UP` übergeben wurden, und zusätzlich auch die umgesetzten ASCII-Zeichen (translated), die sonst von `WM_CHAR` übergeben wurden.

Initialisierung

Vor der Meldungsschleife muß die Anwendung jedoch initialisiert und das Hauptfenster angelegt werden; dies geschieht in der Funktion `SayInitApp`. Die Version für den Presentation Manager beginnt mit zwei neuen Aufrufen: Zunächst wird `WinInitialize` aufgerufen, wodurch das Programm für den Presentation Manager initialisiert wird und eine Handle auf den »Anchor-Block« übergeben wird, die hier `hAB` genannt wird. Der Anchor-Block wird vom Presentation Manager nicht verwendet, er ist nur für die Kompatibilität mit zukünftigen SAA-Umgebungen vorhanden. Der Parameter `hAB` wird jedoch bei mehreren Funktionsaufrufen benötigt.

Die Verwendung von `WinInitialize` bedeutet nicht, daß es sich bei einer Anwendung um eine Presentation-Manager-Anwendung handelt. Tatsächlich können sogar zeichenorientierte OS/2-Anwendungen `WinInitialize` verwenden, wenn sie die lokale Speicherverwaltung oder die Atom-Funktionen des Presentation Managers verwenden wollen. Dann wird `WinCreateMsgQueue` aufgerufen, mit der die Meldungsqueue zugewiesen wird. Anders als unter Windows muß man dies hier selbst machen. Dadurch erhält man eine größere Flexibilität: Man kann die Größe der Meldungsqueue angeben oder einfach die Standardgröße verwenden, indem man als Größe 0 angibt, wie SayWhat das macht.

`WinCreateMsgQueue` *ist* der besondere Aufruf, der besagt, daß es sich um eine Presentation-Manager-Anwendung handelt. Er richtet diesen Thread als Meldungsqueue-Thread ein und bewirkt, daß der Grafikmodus eingestellt wird, wenn eine Presentation-Manager-Anwendung gestartet wird und die übliche Oberfläche anstelle der Presentation Manager-Oberfläche läuft. Wenn Sie mehrere Threads verwenden, können einige Threads eine Meldungsqueue haben und andere nicht, doch jeder Thread, der Fenster anlegen will, muß `WinCreateMsgQueue` aufrufen, da Threads sich nicht eine Queue teilen können.

```
# SW
# MAKE-Datei für SAYWHAT (Windows-Version)

sw.obj:  sw.c  sw.h
    cl -c -AS -DLINT_ARGS -Gcsw -Oas -W3 -Zdp sw.c

sw.res:  sw.rc  sw.h
    rc -r sw.rc

saywhat.exe:  sw.obj  sw.res  sw.def
    link4 sw, saywhat/align:16, saywhat/map/line, slibw, sw.def
    mapsym saywhat
    rc sw.res saywhat.exe
```

Listing 1w: *Die MakeDatei* SW *für SayWhat.*

```
# SWP
# MAKE-Datei für SAYWHAT (Presentation-Manager-Version)

swp.obj:    swp.c  swp.h
    cl -c -AS -DLINT_ARGS -G2csw -Oat -W3 -Zp swp.c

swp.res:    swp.rc  swp.h
    rc -r swp.rc

saywhatp.exe:  swp.obj  swp.res  swp.def
    link @swp.lnk
    mapsym saywhat
    rc swp.res saywhat.exe
```

Listing 1pm: *Die MakeDatei* SWP *für SayWhat.*

Beide Anwendungen laden dann einige String-Ressourcen. Dies ist in den beiden Versionen fast identisch, bis auf den Parameter hAB, der von der Presentation-Manager-Version verwendet wird. Darauf werden einige Informationen über den Bildschirm abgefragt: die Anzahl verfügbarer Farben, die Größe des System-Fonts und die Bildschirmgröße. In der Windows-Version wird ein Informations-Kontext angelegt, der wie ein Einheiten-Kontext ist, jedoch nur zum Sammeln von Informationen über eine Einheit verwendet wird. Dann werden mit GetTextMetrics Informationen über den System-Font und mit GetDeviceCaps die Anzahl der Bildschirmebenen abgefragt.

Die Presentation-Manager-Version ist hier etwas anders. Anstelle eines Informations- oder Einheiten-Kontextes weist sie einen Präsentationsbereich (presentation space) mit WinGetPS zu. Obwohl der Presentation Manager auch Einheiten-Kontexte kennt, verwenden die meisten Grafikfunktionen statt dessen einen Präsentationsbereich, der in einer einfachen Anwendung wie SayWhat genauso verwendet werden kann, wie Kontexte in der Windows-Version. Der Präsentationsbereich ist jedoch wesentlich leistungsfähiger und flexibler; beispielsweise können damit Grafikaufrufe aufgezeichnet und später ausgeführt werden, ähnlich wie bei einer Metadatei. Dadurch kann man es sich sparen, die Meldung WM_PAINT zu bearbeiten. (Dadurch wird die Leistung nicht verbessert, doch es kann bei einigen Anwendungen eine Programmiererleichterung sein.) Die Einzelheiten der Präsentationsbereiche sollen hier nicht beschrieben werden, da SayWhat nichts sonderlich interessantes damit macht.

Nachdem sie eine Handle auf den Präsentationsbereich erhalten hat, ruft die Presentation-Manager-Version GpiQueryCharBox auf, um die Größe der Zeichen des System-Fonts festzustellen, und GpiQueryColorData, um die Anzahl der Farben zu erhalten. Es ist zu beachten, daß hier die Anzahl der wirklich verschiedenen Farben verwendet wird, nicht die Anzahl der Bildschirmebenen wie in der Windows-Version.

Anlegen eines Fensters

Nun kann die Fensterklasse registriert und das Hauptfenster angelegt werden. Hier stößt man auf einen großen Unterschied zwischen Windows und dem Presentation Manager - der Fensteraufbau einer Standardanwendung.

Unter Windows besteht das Fenster einer Standardanwendung aus zwei Teilen: dem Anwendungs-Bereich (client area), auf den ein Programm normalerweise zugreift und der Nicht-Anwendungs-Bereich, der alle Steuerungsmöglichkeiten um das Fenster herum beinhaltet: die Titelzeile, der Rahmen zur Größeneinstellung, das Menü usw. Ihre Fensterfunktion erhält Meldungen für den Anwendungs- und den Nicht-Anwendungs-Bereich, gewöhnlich werden die Meldungen WM_NCxxxx (nonclient) an DefWindowProc weitergegeben. Auch wenn dies bei den meisten Anwendungen funktioniert, hat dieser Aufbau einige Probleme. Das Verhalten der Nicht-Anwendungs-Steuerungen ist direkt in Windows programmiert. Es ist möglich, einige der Verhaltensweisen dadurch zu beeinflussen, daß die Meldungen WM_NCxxxx abgefangen werden, doch dies ist sehr umständlich und ziemlich eingeschränkt. Es gibt auch einige Inkonsistenzen, z.B. die Tatsache, daß die Standard-Scrollbalken Teil des Nicht-Anwendungsbereichs sind, Scrollbalken, die nicht dem Standard entsprechen jedoch als Unterfenster angelegt und anders behandelt werden.

Als ich den Fenstereditor für eine meiner Anwendungen programmierte, war diese Struktur ein ernsthaftes Hindernis. Ich benötigte die volle Kontrolle über alle Aspekte des Verhaltens eines Fensters, da meine Anwendung es dem Anwender ermöglicht, nebenbei die Fenstereigenschaften aufzubauen und zu ändern. Ich wollte nicht, daß sich die Nicht-Anwendungsbereiche im Entwurfs-Modus normal verhalten, da sie zu diesem Zeitpunkt Dinge waren, die editiert werden sollten. Letzendlich gelang es mir, das ganze halbwegs zum Laufen zu bringen, doch nicht genauso, wie ich es mir vorstellte. Wenn ein Benutzer zum Beispiel das Maximize-Sinnbild von einem Fenster entfernen wollte, mußte ich das ganze Fenster löschen und neu anlegen.

```
Keine Link-Datei in der Windows-Version!
```

Listing 2w: *Die Windows-Version benötigt keine Link-Datei.*

```
swp
saywhatp/align:16
saywhatp/map
wincalls doscalls mlibc286/NOD
swp.def
```

Listing 2pm: *Die Link-Datei* SWP.LNK *für SayWhat.*

Beim Presentation Manager wurde dieser Aufbau beseitigt. Die Funktionalität ist noch verfügbar, sie ist jedoch mit Hilfe von Unterfenstern und einigen vordefinierten Fensterklassen realisiert. Alle Dinge, die Teil des Nicht-Anwendungsbereichs waren, sind nun Unterfenster und man kann besonderes mit ihnen anstellen, indem sie einfach als Unterklasse verwendet werden, genauso wie das bei jedem anderen Fenster gemacht wird. Es gibt eine neue Fensterklasse, das Rahmenfenster, das alle diese Unterfenster beinhaltet und sie in Abhängigkeit von der Fenstergröße positioniert. Was unter Windows der Anwendungsbereich war, ist nun selbst ein Unterfenster des Rahmenfensters.

Doch für die meisten Anwendungen macht das keinen Unterschied. Man kann die Funktion WinCreateStdWindow verwenden, um ein Standard-Rahmenfenster mit den gewünschten Steuerungsmöglichkeiten anzulegen, und alles verhält sich im wesentlichen genauso wie vorher. Der einzige gravierende Unterschied besteht darin, daß einige Operationen, die bisher mit Windows-Funktionen durchgeführt wurden, nun dadurch erreicht werden, daß Meldungen an die diversen Steuerungen geschickt werden.

Doch bei Anwendungen, bei denen die Steuerungen im Rahmenfenster besonders behandelt werden, wird einem mit diesem Aufbau das Leben wesentlich leichter gemacht. Die Behandlung von Unterfenstern ist wesentlich einfacher, als die Behandlung aller möglichen WM_NCxxxx-Meldungen; es ist eine übersichtlichere und wesentlich konsistentere Struktur. Man kann nun sogar weitere Steuerungen zum Rahmenfenster hinzufügen und sie beliebig plazieren.

Der Fensteraufbau wurde noch weiter verbessert. Unter Windows hat jedes Fenster ein übergeordnetes Fenster (parent window) und ein Besitzer-Fenster (owner window). Das übergeordnete Fenster bestimmt, wie der Bildschirmplatz verwendet wird - ein Unterfenster steht im übergeordneten Fenster und wird an dessen Rändern abgeschnitten. Der Besitzer bestimmt zwei Dinge: Steuerungsfenster (zum Beispiel Editierfelder) schicken Benachrichtigungsmeldungen an ihren Besitzer und Pop-Up-Fenster verschwinden, wenn der Besitzer zum Sinnbild wird. Dies sind zwei verschiedene Schritte, obwohl Windows sie in einer einzigen Handle für das übergeordnete Fenster kombiniert und diese Handle entsprechend den Stil-Bits des Fensters interpretiert. Für ein WS_CHILD-Fenster ist das übergeordnete Fenster tatsächlich sowohl das übergeordnete Fenster als auch der Besitzer. Für ein WS_POPUP-Fenster ist es NULL und das übergeordnete Fenster ist tatsächlich der Besitzer.

Der Presentation Manager trennt die Handles des übergeordneten und des Besitzer-Fensters, so daß sie individuell angegeben werden können. Dadurch erübrigen sich die Fenstertypen WS_CHILD, WS_POPUP und WS_OVERLAPPED; statt dessen ist ein Fenster ganz einfach nur ein Fenster. Des weiteren hat *jedes* Fenster ein übergeordnetes Fenster, denn es gibt ein neues Fenster, das Desktop-Fenster genannt wird, und das aus dem ganzen Bildschirm besteht. Jedes Fenster der obersten Ebene ist in Wirklichkeit ein Unterfenster des Desktop-Fensters.

Unter Berücksichtigung dieser Dinge wollen wir uns die Aufrufe von WinRegisterClass und WinCreateStdWindow in SayWhat einmal ansehen. Zum Aufruf von WinRegisterClass gibt es nicht viel zu sagen. Die Struktur WNDCLASS aus der Windows-Version ist verschwunden, da die meisten dadurch angegebenen Eigenschaften auf die Klasse des Rahmenfensters und auf WinCreateStdWindow übergegangen sind. Eine interessante Option von WinRegisterClass ist das Stil-Bit CS_SYNCPAINT. Wenn dieses Bit gesetzt ist, werden WM_PAINT-Meldungen dieser Klasse nicht in der üblichen Weise abgesetzt. Statt dessen wird WM_PAINT immer sofort dann geschickt, wenn das Fenster für ungültig erklärt wird (invalidated). Dadurch werden die Rahmensteuerungen sofort neu gezeichnet; sie sind vom Typ CS_SYNCPAINT. In der Windows-Version wird dies von WM_PAINT besonders behandelt, unter dem Presentation Manager ist diese Möglichkeit jedoch jedem Fenster verfügbar. Da die Fensteranzeigeroutine von SayWhat sehr einfach und ziemlich schnell ist, bewirkt CS_SYNCPAINT eine saubere Bildschirmanzeige. Eine Fensterklasse, die länger zur Anzeige benötigt, sollte dieses Stil-Bit vermeiden, dann erhält das Fenster die abgesetzten WM_PAINT-Meldungen genauso, wie unter Windows.

Der Aufruf von WinCreateStdWindow legt das Rahmenfenster von SayWhat sowie das Anwendungsfenster und die Rahmensteuerungen an. Von diesem Aufruf werden die Handles auf das Rahmenfenster und das Anwendungsfenster übergeben, das letztere durch einen Pointer im letzten Parameter. Die Fenster-Stil-Bits sehen ziemlich vertraut aus, doch die meisten beginnen mit FS_ anstatt WS_, da sie wirklich nur Anzeiger sind, die dem Rahmenfenster mitteilen, welches Unterfenster angelegt werden soll. Auch gibt es bei diesem Aufruf keine Angaben über die Position oder Größe. Der Trick besteht darin, daß einem Fenster mit dem Stil WS_VISIBLE automatisch eine passende Standardgröße- und -position zugewiesen wird. Alternativ kann man

```
; SW.DEF
; SW.DEF - Moduldefinitionsdatei für SAYWHAT (Windows-Version)

NAME      SayWhat

DESCRIPTION 'Say What!'

STUB      'WINSTUB.EXE'

CODE      MOVEABLE
DATA      MOVEABLE MULTIPLE

HEAPSIZE  128
STACKSIZE 4096

EXPORTS
    SayAboutDlgProc      @1
    SayWhatDlgProc       @2
    SayWhatWndProc       @3
```

Listing 3w: *Die Modul-Definitionsdatei* SW.DEF *für SayWhat.*

```
; SWP.DEF
; SWP.DEF - Moduldefinitionsdatei für SAYWHAT (PM-Version)

NAME      SayWhat

DESCRIPTION 'Say What!'

STUB      'OS2STUB.EXE'

CODE      MOVEABLE
DATA      MOVEABLE MULTIPLE

HEAPSIZE  128
STACKSIZE 4096

EXPORTS
    SayAboutDlgProc      @1
    SayWhatDlgProc       @2
    SayWhatWndProc       @3
```

Listing 3pm: *Die Modul-Definitionsdatei* SWP.DEF *für SayWhat.*

es unsichbar anlegen, es dann mit WinSetWindowPos explizit an eine gewünschte Stelle positionieren und dann sichtbar machen.

WinCreateStdWindow ist nur als bequeme Methode zur Anlage eines Standard-Rahmenfensters gedacht. Es gibt auch eine allgemein verwendbares WinCreateWindow, mit dem jede Art von Fenster angelegt werden kann, wobei man die totale Kontrolle darüber hat, wie das Fenster aufgebaut ist. Diesen Aufruf wird man verwenden, um zusätzliche Unterfenster oder jede andere Art von besonderem Fenster anzulegen.

Fensterfunktionen

Jetzt, wo das Fenster angelegt ist, ist es Zeit, sich die Fensterfunktion SayWhatWndProc genauer anzuschauen. Es ist zu sehen, daß die Fensterfunktionen in beiden Versionen sehr ähnlich sind. Wie ich bereits erwähnt habe, erfolgen alle Tastatureingaben unter der Meldung WM_CHAR, nicht bei WM_KEYDOWN, wie in der Windows-Version; doch ansonsten sind beide im wesentlichen gleich. Die Mauseingaben sind ebenfalls fast identisch, nur die Namen sind geändert worden, damit die Inkonsistenz vermieden wird, daß der rechte Mausknopf die Meldungen WM_LBUTTONUP/DOWN schickt, wenn die Knöpfe vertauscht worden sind, der Hauptmausknopf heißt statt dessen WM_BUTTON1.

Auch die anderen Meldungen, die von SayWhatWndProc behandelt werden, werden sehr ähnlich verarbeitet: WM_CREATE, WM_DESTROY, WM_SIZE, WM_PAINT, WM_TIMER und WM_COMMAND. Ein sofort ins Auge fallender Unterschied ist, daß es kein entsprechendes wParam == SIZEICONIC gibt, doch es gibt eine neue Meldung WM_MINMAX, die die gleichen Informationen bereitstellt. WM_MINMAX wird also abgefangen und bearbeitet um herauszufinden, wann das Fenster zum Sinnbild gemacht oder wieder auf die normale Größe wiederhergestellt wird.

Es gibt jedoch bei allen Meldungen einige Unterschiede bei den Parametern. Anstelle eines 16-Bit-Parameters wParam und eines 32-Bit-Parameters lParam sind nun zwei 32-Bit-Parameter vorhanden: lParam1 und lParam2. Dadurch können mit Meldungen etwas mehr Informationen übergeben werden, doch der eigentliche Grund für diese Änderung war die Berücksichtigung solcher Meldungen, die eine zusätzliche Fenster-Handle in wParam übergeben, wie zum Beispiel WM_VSCROLLCLIPBOARD. Unter dem Presentation Manager sind Fenster-Handles und die meisten anderen Handles 32 Bit groß, nicht 16 Bit, wParam mußte also erweitert werden, um dies zu ermöglichen. Der Parameter hWnd der Fensterfunktion ist ebenfalls ein 32-Bit-Parameter. Achten Sie darauf, wenn sie Programme haben, die davon ausgehen, daß Handles 16 Bit groß sind.

Fensteranzeige

Die Fensteranzeige wird in beiden Versionen von der Funktion SayWhatPaint ausgeführt. Sie ist in beiden Versionen sehr ähnlich, obwohl die Presentation-Manager-Version keine PAINTSTRUCT hat. Statt dessen übergibt die Funktion WinBeginPaint eine Handle auf einen Präsentationsbereich und das zu aktualisierende Rechteck als separaten Parameter. Des weiteren verwendet es GpiSetColor anstelle von SetTextColor und GpiCharStringAt anstatt TextOut. In beiden Fällen werden jedoch die gleichen Aufgaben erledigt.

Es gibt einen besonderen Effekt in der Anzeigelogik von SayWhat. Wenn Sie SayWhat laufenlassen, werden Sie bemerken, daß der Text sich sauber auf dem Bildschirm umherbewegt. Es gibt kein Flimmern, obwohl der Text jedesmal neu angezeigt wird. Bei vielen Windows-Anwendungen macht sich bei der Textanzeige ein leichtes Flimmern bemerkbar, was dadurch verursacht wird, das meistens der Hintergrund komplett gelöscht wird und anschließend der

```
#include <style.h>
#include "sw.h"

STRINGTABLE
BEGIN
    STR_NAME,     "SayWhat!"
    STR_TITLE,    "Say What!"
    STR_WHAT,     "Hello Windows!"
END


SayWhat!    MENU
BEGIN

  POPUP "&Say"
  BEGIN
    MENUITEM "&What..."                  , CMD_WHAT
    MENUITEM SEPARATOR
    MENUITEM "E&xit"                     , CMD_EXIT
    MENUITEM SEPARATOR
    MENUITEM "A&bout SayWhat!..."        , CMD_ABOUT
  END

END


DLG_ABOUT DIALOG 19, 17, 130, 83
STYLE WS_DLGFRAME | WS_POPUP
BEGIN
  CTEXT "Microsoft Windows",    -1,  0,  8, 127,  8
  CTEXT "Say What!",            -1,  0, 18, 127,  8
  CTEXT "Version 1.00",         -1,  0, 30, 127,  8
  CTEXT "By Michael Geary",     -1,  0, 44, 129,  8
  DEFPUSHBUTTON "Ok",         IDOK, 48, 62,  32, 14
END


DLG_WHAT DIALOG 49, 41, 177, 103
CAPTION "Say What!"
STYLE WS_CAPTION | WS_SYSMENU | WS_VISIBLE | WS_POPUP
BEGIN
  CONTROL "Say &What:"  -1,                 "Static",   SS_LEFT |\
  WS_GROUP,                                    8, 10,  40,  8
  CONTROL ""            ITEM_WHAT,          "Edit",     ES_LEFT |\
  ES_AUTOHSCROLL | WS_BORDER | WS_TABSTOP,  56,  8, 112, 12
  CONTROL "&Time Delay:" -1,                "Static",   SS_LEFT,\
                                               8, 28,  53,  8
  CONTROL ""            ITEM_INTBAR,        "ScrollBar",SBS_HORZ |\
  WS_TABSTOP,                                 56, 28,  88,  8
  CONTROL ""            ITEM_INTERVAL,      "Edit",     ES_LEFT |\
  WS_BORDER | WS_TABSTOP,                    152, 26,  16, 12
  CONTROL "&Stability:"  -1,                "Static",   SS_LEFT,\
                                               8, 46,  56,  8
  CONTROL ""            ITEM_DISTBAR,       "ScrollBar",SBS_HORZ |\
  WS_TABSTOP,                                 56, 46,  88,  8
  CONTROL ""            ITEM_DISTANCE,      "Edit",     ES_LEFT |\
  WS_BORDER | WS_TABSTOP,                    152, 44,  16, 12
  CONTROL "Painting:"    -1,                "Static",   SS_LEFT,\
                                               8, 64,  40,  8
  CONTROL "&Clean"       ITEM_CLEAN,        "Button",   \
  BS_AUTORADIOBUTTON | WS_GROUP | WS_TABSTOP,56,62,  36, 12
  CONTROL "&Flicker"     ITEM_FLICKER,      "Button",   \
  BS_AUTORADIOBUTTON,                         96, 62, 42, 12
  CONTROL "Enter"        IDOK,              "Button",   \
  BS_DEFPUSHBUTTON | WS_GROUP | WS_TABSTOP,  24, 82,  48, 14
  CONTROL "Esc=Close"    IDCANCEL,          "Button",   \
  BS_PUSHBUTTON | WS_TABSTOP,                104, 82,  48, 14
END
```

Listing 4w: *Die Ressource-Datei* `SW.RC` *für SayWhat (\ am Ende der Zeile = folgende Zeile gehört dazu).*

```
#include <style.h>
#include "swp.h"

STRINGTABLE
{
  STR_NAME,     "SayWhat!"
  STR_TITLE,    "Say What!"
  STR_WHAT,     "Hello Windows!"
}


MENU SayWhat!
{
  SUBMENU "~Say",                     -1
  {
    MENUITEM "~What...",              CMD_WHAT
    MENUITEM SEPARATOR
    MENUITEM "E~xit",                 CMD_EXIT
    MENUITEM SEPARATOR
    MENUITEM "A~bout SayWhat!...",    CMD_ABOUT
  }
}


DLGTEMPLATE DLG_ABOUT
{
  DIALOG "", DLG_ABOUT, 19, 17, 130, 83, \
  FS_DLGBORDER | WS_SAVEBITS | WS_VISIBLE
  {
    CTEXT "Microsoft Windows",    -1,  0,  8, 127,  8
    CTEXT "Say What!",            -1,  0, 18, 127,  8
    CTEXT "Version 1.00",         -1,  0, 30, 127,  8
    CTEXT "By Michael Geary",     -1,  0, 44, 129,  8
    DEFPUSHBUTTON "Ok",         IDOK, 48, 62,  32, 14
  }
}


DLGTEMPLATE DLG_WHAT
{
  DIALOG "Say What!", DLG_WHAT, 49, 41, 177, 103, \
  FS_TITLEBAR | FS_SYSMENU | FS_BORDER | WS_VISIBLE
  {
    CONTROL "Say ~What:"   -1,                8, 10,  40,  8, \
    WC_STATIC,     SS_LEFT | WS_GROUP
    CONTROL ""             ITEM_WHAT,        56,  8, 112, 12, \
    WC_EDIT,   ES_LEFT | ES_AUTOSCROLL | ES_MARGIN | WS_TABSTOP
    CONTROL "~Time Delay:" -1,                8, 28,  53,  9, \
    WC_STATIC,     SS_LEFT
    CONTROL ""             ITEM_INTBAR,      56, 28,  88,  8, \
    WC_SCROLLBAR, SBS_HORZ | WS_TABSTOP
    CONTROL ""             ITEM_INTERVAL,   152, 26,  16, 12, \
    WC_EDIT,       ES_LEFT | ES_MARGIN | WS_TABSTOP
    CONTROL "~Stability:"  -1,                8, 46,  56,  9, \
    WC_STATIC,     SS_LEFT
    CONTROL ""             ITEM_DISTBAR,     56, 46,  88,  8, \
    WC_SCROLLBAR, SBS_HORZ | WS_TABSTOP
    CONTROL ""             ITEM_DISTANCE,   152, 44,  16, 12, \
    WC_EDIT,       ES_LEFT | ES_MARGIN | WS_TABSTOP
    CONTROL "Painting:"    -1,                8, 64,  40,  8, \
    WC_STATIC,     SS_LEFT
    CONTROL "~Clean"       ITEM_CLEAN,       56, 62,  36, 12, \
    WC_BUTTON,     BS_AUTORADIOBUTTON | WS_GROUP | WS_TABSTOP
    CONTROL "~Flicker"     ITEM_FLICKER,     96, 62,  42, 12, \
    WC_BUTTON,     BS_AUTORADIOBUTTON
    CONTROL "Enter"        IDOK,             24, 82,  48, 14, \
    WC_BUTTON,     BS_DEFPUSHBUTTON | WS_GROUP | WS_TABSTOP
    CONTROL "Esc=Close"    IDCANCEL,        104, 82,  48, 14, \
    WC_BUTTON,     BS_PUSHBUTTON | WS_TABSTOP
  }
}
```

Listing 4pm: *Die Ressource-Datei* `SWP.RC` *für SayWhat (\ am Ende der Zeile = folgende Zeile gehört dazu).*

neue Text angezeigt wird. Das ist in Ordnung, wenn das Fenster das erste Mal angezeigt wird, wenn jedoch ein Teil des Fensters neu angezeigt werden soll, gibt es einen Sekundenbruchteil, in dem der Hintergrund leer ist.

Dieser Effekt ist Ihnen sicherlich schon beim Windows-Notizblock und in Write aufgefallen. Wenn Sie in einem dieser Programme sehr schnell eingeben, flimmert die Zeile, in der die Eingabe erfolgt, ein wenig. In SayWhat können Sie dieses Problem dadurch sichtbar machen, daß Sie die Option »Flicker« in der Steuerungs-Dialogbox einschalten. Es ist dann sogar noch schlimmer, weil das Fenster so oft neu angezeigt wird.

Wie vermeidet SayWhat diese Flimmern nun im normalen Modus? Ganz einfach: Es löscht den Hintergrundtext unter dem anzuzeigenden Text nicht. Es löscht wenn es nötig ist alle Hintergrundinformationen, die nicht neu angezeigt werden, doch dort, wo neuer Text angezeigt wird, wird nur der Text geschrieben ohne vorher zu löschen. Dies funktioniert, weil Text normalerweise nicht durchsichtig ausgegeben wird; er überdeckt komplett das, was darunter liegt.

Es gibt in SayWhat zwei verschiedene Anzeigeroutinen, um dies zu demonstrieren. Wenn die Variable `bCleanPaint` `FALSE` ist, wird die übliche, flimmernde Methode verwendet, das heißt, zunächst wird der ganze Hintergrund gelöscht und dann der Text angezeigt. Wenn `bCleanPaint` `TRUE` ist, wird der Text ausgegeben und nur die übrigen Teile des Hintergrunds gelöscht.

Es mag seltsam erscheinen, daß ich mich über ein so kleines Problem so breit auslasse, doch die Korrektur dieses kleinen Problems verbessert die Erscheinung einer Anwendung wesentlich. Ich weiß dies aus eigener Erfahrung: Der Gliederungseditor in meiner eigenen Anwendung flimmerte in der ersten Version sehr stark und irritierte mich dadurch bei der Arbeit sehr. Dies verschwand, als ich es änderte, und diese Methode zur sauberen Anzeige verwendete. Und wenn es auch etwas zusätzlichen Programmcode in der Anzeigefunktion erforderte, wurde dadurch die Gesamtgröße und die Komplexität des Programms verringert. Das Flimmern war so störend, daß ich schon alle möglichen Sonderfälle berücksichtigen wollte, um sicher zu gehen, daß ich niemals mehr Fensterbereiche ungültig mache, als unbedingt notwendig war. Mit dieser sauberen Anzeigemethode ist das nun gleichgültig. Die Anzeige erfolgt nun ganz unaufdringlich, wenn ich also ein wenig zu viel für ungültig erkläre, macht das nichts.

Dialogboxen

SayWhat hat zwei Dialogboxen: eine modale About-Box und ein modusloses Steuerungfeld, die beide unter dem Fall `WM_COMMAND` in `SayWhatWndProc` behandelt werden. Der Programmcode ist in beiden Versionen ähnlich, bis auf die Tatsache, daß `MakeProcInstance` in der geschützten Umgebung von OS/2 nicht benötigt wird. OS/2 und die Speicherverwaltungshardware weisen die korrekten Speicherbereiche für Dialogboxen automatisch zu. Dies ist ein Segen für alle, die sich öfter mit seltsamen Abstürzen unter Windows herumschlagen mußten, weil sie `MakeProcInstance` vergessen hatten. Diese Funktionen und die Fensterfunktionen müssen jedoch wie zuvor mit `EXPORT` deklariert werden.

Es gibt einen wichtigen Unterschied in der Implementation der Dialogbox-Funktion selbst. Unter Windows wird eine Dialogbox-Funktion nicht wie eine normale Fensterfunktion programmiert. Die Fensterfunktion für alle Dialogboxen ist eine interne Funktion innerhalb von Windows mit dem Namen `DlgWndProc`, die Ihre Dialogfunktion aufruft, bevor sie ihre eigene Meldungsbearbeitung durchführt; wenn die Dialogfunktion einen Wert ungleich Null zurückgibt, wird die eigene Meldungsbearbeitung übergangen. Ein Problem bei dieser Vorgehensweise besteht darin, daß Dialogfunktionen anders arbeiten, als normale Fensterfunktionen. Ein anderes Problem ist, daß es für eine Dialogfunktion unmöglich ist, einen Wert an den `SendMessage`-Aufruf zurückzugeben. Der Rückgabewert wird nicht an `SendMessage` weitergegeben, `DlgWndProc` übergibt einfach Null. (Um das ganze noch unübersichtlicher zu machen, wird die Meldung `WM_CTLCOLOR` besonders behandelt und dabei der übergebene Wert zurückgegeben.)

Unter dem Presentation Manager funktionieren Dialogfunktionen genauso wie jede andere Fensterfunktion. Ihr Rückgabewert ist der wirkliche Rückgabewert und es gibt eine Standard-Meldungsfunktion `WinDefDlgProc`, die für Meldungen aufgerufen wird, die man nicht selbst bearbeitet. `SayAboutDlgProc` zeigt dies und macht, wie die meisten About-Boxen nicht mehr, als sich selbst zu schließen, wenn der OK-Button betätigt wird.

Die andere Dialogbox, das Steuerungsfeld von SayWhat, ist interessanter. Diese Dialogbox hat kein Besitzerfenster (unter Windows übergeordnetes Fenster). Normalerweise wird ein Besitzerfenster angegeben, wenn eine Dialogbox angelegt wird, was drei Dinge bewirkt: Die Box wird relativ zum übergeordneten Fenster positioniert, sie liegt immer über dem übergeordneten Fenster und sie verschwindet, wenn das übergeordnete Fenster zum Sinnbild gemacht wird. In SayWhat wollte ich die beiden letzten Dinge jedoch nicht. Ich wollte, daß die Box sichtbar bleibt, wenn das SayWhat-Fenster zum Sinnbild gemacht wird, so daß man mit den Paramtern spielen kann, während das Sinnbild angezeigt wird. Ich wollte auch das Hauptfenster von SayWhat über die Dialogbox legen können.

Dies wird dadurch ermöglicht, daß als Besitzer `NULL` angegeben wird. Ich wollte jedoch, daß die Box weiterhin relativ zum Hauptfenster positioniert wird und nicht relativ zum ganzen Bildschirm. Die Positionierung wird also im Programmcode unter `WM_INITDIALOG` explizit durchgeführt, indem in der Windows-Version `ClientToScreen` und in der Presentation-Manager-Version `WinMapWindowPoints` verwendet wird. `WinMapWindowPoints` ist

```
/* SW.H - C-Headerdatei für SAYWHAT (Windows-Version) */

/* Konstanten für Stringtabelle */

#define STR_NAME        101
#define STR_TITLE       102
#define STR_WHAT        103

/* IDs für Menübefehle */

#define CMD_ABOUT       201
#define CMD_EXIT        202
#define CMD_WHAT        203

/* IDs der Dialogbox-Ressourcen */

#define DLG_ABOUT       301
#define DLG_WHAT        302

/* 'What...' IDs der Dialogboxfelder */

#define ITEM_WHAT       401
#define ITEM_INTBAR     402
#define ITEM_INTERVAL   403
#define ITEM_DISTBAR    404
#define ITEM_DISTANCE   405
#define ITEM_CLEAN      406
#define ITEM_FLICKER    407

/* Timer IDs */

#define TIMER_MOVE      501
#define TIMER_CHAR      502
```

Listing 5w: *Die Header-Datei* `SW.H` *für SayWhat.*

```
/* SWP.H - C-Headerdatei für SAYWHAT (PM-Version) */

/* Stringtabellenkonstanten */

#define STR_NAME        101
#define STR_TITLE       102
#define STR_WHAT        103

/* IDs Befehlsmenüs */

#define CMD_ABOUT       201
#define CMD_EXIT        202
#define CMD_WHAT        203

/* IDs Dialogboxressourcen */

#define DLG_ABOUT       301
#define DLG_WHAT        302

/* 'What...' IDs Dialogboxfelder */

#define ITEM_WHAT       401
#define ITEM_INTBAR     402
#define ITEM_INTERVAL   403
#define ITEM_DISTBAR    404
#define ITEM_DISTANCE   405
#define ITEM_CLEAN      406
#define ITEM_FLICKER    407

/* Timer IDs */

#define TIMER_MOVE      501
#define TIMER_CHAR      502

/* IDs Menüressource */

#define MENU_WHAT       601
```

Listing 5pm: *Die Header-Datei* `SWP.H` *für SayWhat.*

eine nützliche Funktion, die `ClientToScreen` und `ScreenToClient` ersetzt, denn es kann ein Quellfenster, ein Zielfenster oder beides angegeben werden. Es setzt auch mit einem Aufruf so viele Punkte um, wie Sie möchten. In diesem Programm werden zwei Punkte umgesetzt, das heißt, ein Rechteck.

Nach der Umsetzung müssen Sie sich vergewissern, daß die Dialogbox nicht teilweise außerhalb des Bildschirms positioniert wurde, man muß es also gegen die Bildschirmmaße gegenprüfen und wenn nötig entsprechend anpassen. Dann wird mit `SetWinPos` oder `WinSetWindowPos` die neue Position eingestellt. Diese Funktion arbeitet unter dem Presentation Manager etwas anders. Wenn Sie `SetWindowPos` unter Windows 2.0 bereits verwendet haben, dann wissen Sie, daß sie eine Reihe von Dingen gleichzeitig erledigen kann, zum Beispiel das Fenster bewegen, vergrößern oder die Fensterreihenfolge verändern. Die Standardeinstellung ist, daß alles gleichzeitig gemacht wird und der letzte Parameter gibt an, was die Funktion nicht machen soll. `WinSetWindowPos` macht das gleiche, nur der letzte Parameter gibt an, was gemacht werden soll, also genau umgekehrt.

Bitte beachten Sie, daß der y-Parameter von `WinSetWindowPos` die Fensterunterkante ist, nicht die Oberkante. Hier zeigt sich ein kleiner Unterschied zwischen Windows und dem Presentation Manager: Vertikale Koordinaten laufen von unten nach oben, nicht von oben nach unten, das bedeutet, die Position (0,0) ist die linke untere Ecke eines Fensters oder des Bildschirms, nicht die linke obere Ecke. Um damit übereinzustimmen, wird eine Rechteckstruktur nun in der Reihenfolge links, unten, rechts, oben anstatt links, oben, rechts, unten gespeichert. Wenn Sie mit den verschiedenen GDI-Umsetzungsmodi vertraut sind, die in Windows verfügbar sind, wird Ihnen sicherlich aufgefallen sein, daß sie bis auf `MM_TEXT` alle von unten nach oben laufen; bei `MM_TEXT` laufen die Koordinaten von oben nach unten. Durch die Änderung im Presentation Manager wird diese Inkonsistenz beseitigt, alle Koordinaten laufen normalerweise in die gleiche Richtung, dies erfordert jedoch einigen Umsetzungsaufwand. Man kann dem Presentation Manager sagen, daß er in einem Fenster die Koordinatenrichtung von oben nach unten verwenden soll, so daß dadurch die Umsetzung bestehender Anwendungen erleichtert wird.

```
/*
 * SW.C - C-Code für SayWhat - Windows-Version
 */

/* Wichtig: Dieses Programm *muß* mit -Gc kompiliert werden */

#ifndef LINT_ARGS
#define LINT_ARGS  /* Argumentüberprüfung */
#endif

#include <windows.h>
#include <limits.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "sw.h"


#define MIN_INTERVAL    1   /* Grenzen für nInterval */
#define MAX_INTERVAL    999

#define MIN_DISTANCE    1   /* Grenzen für nDistance */
#define MAX_DISTANCE    99



/*  statische Variablen  */

HANDLE  hInstance;              /* Vorkommen-Handle */

HWND    hWndWhat;               /* Handle des Hauptfensters */
HWND    hWndPanel;              /* Handle für Steuerungsfeld */

FARPROC lpfnDlgProc;            /* ProcInstance für Dialog */

char    szAppName[10];          /* Fensterklassenname */
char    szTitle[15];            /* Titel des Hauptfensters */

char    szText[40];             /* aktueller Text */
NPSTR   pText = szText;         /* Zeiger in szText wenn Sinnbild*/
char    cBlank = ' ';           /* ein Leerzeichen zum Zeigen */

RECT    rcText;                 /* aktuelles Text-Rechteck */
POINT   ptAdvance;              /* Schrittweite f. SayAdvanceText*/
POINT   ptCharSize;             /* X- und Y-Größe eines Zeichens */
POINT   ptScreenSize;           /* X- und Y-Größe des Bildschirms*/

int     nDisplayPlanes;         /* Anzahl Bildschirmebenen */
DWORD   rgbTextColor;           /* aktuelle Textfarbe */
HBRUSH  hbrBkgd;                /* für Löschen des Hintergrunds */

int     nInterval = 40;         /* aktueller Wert "Interval" */
int     nDistance = 30;         /* aktueller Wert "Distance" */
int     nDistLeft = 0;          /* Richtung ändern wenn 0 */
BOOL    bCleanPaint = TRUE;     /* sauber/flimmernd anzeigen? */
BOOL    bMouseDown = FALSE;     /* Maus gedrückt? */
BOOL    bIconic = FALSE;        /* Hauptfenster ein Sinnbild? */


/*  komplette Prototypen für die Funktionen  */

BOOL FAR SayAboutDlgProc( HWND, unsigned, WORD, LONG );
void     SayAdvanceTextChar( HWND );
void     SayAdvanceTextPos( HWND );
void     SayChangeColor( HWND );
void     SayDoBarMsg( HWND, HWND, WORD, int );
void     SayFillRect( HDC, int, int, int, int );
void     SayInitBar( HWND, int, int, int, int );
BOOL     SayInitApp( HANDLE, int );
void     SayInvalidateText( HWND );
void     SayLimitTextPos( HWND );
void     SayMoveText( HWND, POINT );
void     SaySetBar( HWND, int *, int );
void     SayExitApp( int );
BOOL FAR SayWhatDlgProc( HWND, unsigned, WORD, LONG );
void     SayWhatPaint( HWND );
LONG FAR SayWhatWndProc( HWND, unsigned, WORD, LONG );
void     WinMain( HANDLE, HANDLE, LPSTR, int );
```

```
/*
 * SWP.C - C-Code für SayWhat - Presentation-Manager-Version
 */

#ifndef LINT_ARGS
#define LINT_ARGS  /* Argumentüberprüfung */
#endif

#include <os2pm.h>
#include <limits.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "swp.h"


#define MIN_INTERVAL    1   /* Grenzen für nInterval */
#define MAX_INTERVAL    999

#define MIN_DISTANCE    1   /* Grenzen für nDistance */
#define MAX_DISTANCE    99

#define COLORDATAMAX    5



/*  statische Variablen  */

HAB     hAB = NULL;             /* Handle Anchorblock */
HMQ     hMsgQ = NULL;           /* Handle Meldungsqueue */

HWND    hWndWhatFrame;          /* Handle Rahmenfenster */
HWND    hWndWhat;               /* Handle Anwendungsfenster */
HWND    hWndPanel;              /* Handle für Steuerungsfeld */

CHAR    szAppName[10];          /* Fensterklassenname */
CHAR    szTitle[15];            /* Titel des Hauptfensters */

CHAR    szText[40];             /* aktueller Text */
NPSZ    npszText = szText;      /* Zeiger in szText wenn Sinnbild*/
CHAR    cBlank = ' ';           /* ein Leerzeichen zum Zeigen */

RECT    rcText;                 /* aktuelles Text-Rechteck */
POINT   ptAdvance;              /* Schrittweite f. SayAdvanceText*/
POINT   ptCharSize;             /* X- und Y-Größe eines Zeichens */
POINT   ptScreenSize;           /* X- und Y-Größe des Bildschirms*/

LONG    lColorMax;              /* Anzahl verfügbarer Farben */
LONG    lColor;                 /* Index der aktuellen Textfarbe */

SHORT   nInterval = 40;         /* aktueller Wert "Interval" */
SHORT   nDistance = 30;         /* aktueller Wert "Distance" */
SHORT   nDistLeft = 0;          /* Richtung ändern wenn 0 */
BOOL    bCleanPaint = TRUE;     /* sauber/flimmernd anzeigen? */
BOOL    bMouseDown = FALSE;     /* Maus gedrückt? */
BOOL    bIconic = FALSE;        /* Hauptfenster ein Sinnbild? */


/*  komplette Prototypen für die Funktionen  */

ULONG FAR PASCAL SayAboutDlgProc( HWND, USHORT, ULONG, ULONG );
VOID         SayAdvanceTextChar( HWND );
VOID         SayAdvanceTextPos( HWND );
VOID         SayChangeColor( HWND );
VOID         SayDoBarMsg( HWND, USHORT, USHORT, SHORT );
VOID         SayExitApp( INT );
VOID         SayFillRect( HPS, SHORT, SHORT, SHORT, SHORT );
VOID         SayInitBar( HWND, SHORT, SHORT, SHORT, SHORT );
BOOL         SayInitApp( VOID );
VOID         SayInvalidateText( HWND );
VOID         SayLimitTextPos( HWND );
VOID         SayMoveText( HWND, POINT );
VOID         SaySetBar( HWND, SHORT *, SHORT );
ULONG FAR PASCAL SayWhatDlgProc( HWND, USHORT, ULONG, ULONG );
VOID         SayWhatPaint( HWND );
ULONG FAR PASCAL SayWhatWndProc( HWND, USHORT, ULONG, ULONG );
void  cdecl  main( INT, PSZ );
```

```
/*  Dialogfunktion für die "About"-Box  */

BOOL FAR SayAboutDlgProc( hWndDlg, wMsg, wParam, lParam )
    HWND        hWndDlg;
    unsigned    wMsg;
    WORD        wParam;
    LONG        lParam;
{
    switch( wMsg )
    {
      case WM_COMMAND:
        switch( wParam )
        {
          case IDOK:
            EndDialog( hWndDlg, TRUE );
            return TRUE;
        }
    }
    return FALSE;
}

/*  Geht im Sinnbildmodus zum nächsten Zeichen.
 *  Nimmt am Ende des Strings ein Leerzeichen.
 */

void SayAdvanceTextChar( hWnd )
    HWND        hWnd;
{
    if( ! bIconic )
        return;
    if( pText == &cBlank )
        pText = szText;
    else if( ! *(++pText) )
        pText = &cBlank;

    SayChangeColor( hWnd );
    SayInvalidateText( hWnd );
}

/*  Ändert die Textposition entsprechend ptAdvance. Vermindert
 *  zunächst nDistLeft, stellt ptAdvance und nDistLeft zufällig
 *  neu ein wenn es Null wird, und ändert die Farbe.
 *  Macht nichts, wenn die Maus betätigt wird, der Text folgt
 *  also der Maus.
 */

void SayAdvanceTextPos( hWnd )
    HWND        hWnd;
{
    int         i;

    if( bMouseDown )
        return;
    SayInvalidateText( hWnd );
    if( nDistLeft-- < 0 ) {
        nDistLeft = rand() % nDistance + 1;
        do {
            i = rand();
            ptAdvance.x = (
                i < SHRT_MAX/3   ? -1
              : i < SHRT_MAX/3*2 ?  0
              :                     1
            );
            i = rand();
            ptAdvance.y = (
                i < SHRT_MAX/3   ? -1
              : i < SHRT_MAX/3*2 ?  0
              :                     1
            );
        } while( ptAdvance.x == 0  &&  ptAdvance.y == 0 );
        if( ! bIconic )
            SayChangeColor( hWnd );
    } else {
        rcText.left   += ptAdvance.x;
        rcText.right  += ptAdvance.x;
        rcText.top    += ptAdvance.y;
        rcText.bottom += ptAdvance.y;
    }
    SayInvalidateText( hWnd );
}
```

```
/*  Dialogfunktion für die "About"-Box  */

ULONG FAR PASCAL SayAboutDlgProc(hWndDlg,wMsg, lParam1,lParam2)
    HWND        hWndDlg;
    USHORT      wMsg;
    ULONG       lParam1;
    ULONG       lParam2;
{
    switch( wMsg )
    {
      case WM_COMMAND:
        switch( LOUSHORT(lParam1) )
        {
          case IDOK:
            WinDismissDlg( hWndDlg, TRUE );
            break;
        }
    }
    return WinDefDlgProc( hWndDlg, wMsg, lParam1, lParam2 );
}

/*  Geht im Sinnbildmodus zum nächsten Zeichen.
 *  Nimmt am Ende des Strings ein Leerzeichen.
 */

VOID SayAdvanceTextChar( hWnd )
    HWND        hWnd;
{
    if( ! bIconic )
        return;
    if( npszText == &cBlank )
        npszText = szText;
    else if( ! *(++npszText) )
        npszText = &cBlank;

    SayChangeColor( hWnd );
    SayInvalidateText( hWnd );
}

/*  Ändert die Textposition entsprechend ptAdvance. Vermindert
 *  zunächst nDistLeft, stellt ptAdvance und nDistLeft zufällig
 *  neu ein wenn es Null wird, und ändert die Farbe.
 *  Macht nichts, wenn die Maus betätigt wird, der Text folgt
 *  also der Maus.
 */

VOID SayAdvanceTextPos( hWnd )
    HWND        hWnd;
{
    SHORT       i;

    if( bMouseDown )
        return;
    SayInvalidateText( hWnd );
    if( nDistLeft-- < 0 ) {
        nDistLeft = rand() % nDistance + 1;
        do {
            i = rand();
            ptAdvance.x = (
                i < SHRT_MAX/3   ? -1
              : i < SHRT_MAX/3*2 ?  0
              :                     1
            );
            i = rand();
            ptAdvance.y = (
                i < SHRT_MAX/3   ? -1
              : i < SHRT_MAX/3*2 ?  0
              :                     1
            );
        } while( ptAdvance.x == 0  &&  ptAdvance.y == 0 );
        if( ! bIconic )
            SayChangeColor( hWnd );
    } else {
        rcText.xLeft   += ptAdvance.x;
        rcText.xRight  += ptAdvance.x;
        rcText.yTop    += ptAdvance.y;
        rcText.yBottom += ptAdvance.y;
    }
    SayInvalidateText( hWnd );
}
```

```
/*  Ändert die Farbe zufällig, wenn die Farbe verfügbar ist.
 *  Die Farbänderung wird erzwungen: Wenn die Zufallsfarbe die
 *  gleich ist wie die alte, wird eine neue ausgewählt.
 */

void SayChangeColor( hWnd )
    HWND        hWnd;
{
    HDC         hDC;
    DWORD       rgbNew;
    DWORD       rgbWindow;

    if( nDisplayPlanes <= 1 ) {
        rgbTextColor = GetSysColor(COLOR_WINDOWTEXT);
    } else {
        rgbWindow = GetSysColor(COLOR_WINDOW);
        hDC = GetDC( hWnd );
        do {
            rgbNew = GetNearestColor(
                hDC,
                MAKELONG( rand(), rand() ) & 0x00FFFFFFL
            );
        } while(rgbNew == rgbWindow || rgbNew == rgbTextColor);
        rgbTextColor = rgbNew;
        ReleaseDC( hWnd, hDC );
    }
}


/*  Behandelt Scrollbalken-Meldungen von der Steuerungs-
 *  Dialogbox. Korrigiert die Scrollbalkenposition, wobei die
 *  Grenzen berücksichtigt werden, kopiert den Scrollbalkenwert
 *  in das daneben liegende Editierfeld und stellt dann
 *  nDistance oder nInterval entsprechend ein.
 */

void SayDoBarMsg( hWndDlg, hWndBar, wCode, nThumb )
    HWND        hWndDlg;
    HWND        hWndBar;
    WORD        wCode;
    int         nThumb;
{
    int         nPos;
    int         nOldPos;
    int         nMin;
    int         nMax;
    int         idBar;

    idBar = GetWindowWord( hWndBar, GWW_ID );

    nOldPos = nPos = GetScrollPos( hWndBar, SB_CTL );
    GetScrollRange( hWndBar, SB_CTL, &nMin, &nMax );


    switch( wCode )
    {
        case SB_LINEUP:         --nPos;         break;

        case SB_LINEDOWN:       ++nPos;         break;

        case SB_PAGEUP:         nPos -= 10;     break;

        case SB_PAGEDOWN:       nPos += 10;     break;

        case SB_THUMBPOSITION:
        case SB_THUMBTRACK:     nPos = nThumb;  break;

        case SB_TOP:            nPos = nMin;    break;

        case SB_BOTTOM:         nPos = nMax;    break;
    }
```

```
/*  Ändert die Farbe zufällig, wenn die Farbe verfügbar ist.
 *  Die Farbänderung wird erzwungen: Wenn die Zufallsfarbe die
 *  gleich ist wie die alte, wird eine neue ausgewählt.
 */

VOID SayChangeColor( hWnd )
    HWND        hWnd;
{
    HPS         hPS;
    LONG        lWindow;
    LONG        lNew;

    hPS = WinGetPS( hWnd );

    if( lColorMax <= 2 ) {
        lColor = GpiQueryColorIndex(
            hPS,
            WinQuerySysColor( hAB, SCLR_WINDOWTEXT ),
            0L
        );
    } else {
        lWindow = GpiQueryColorIndex(
            hPS,
            WinQuerySysColor( hAB, SCLR_WINDOW ),
            0L
        );
        do {
            lNew = rand() % lColorMax;
        } while( lNew == lWindow || lNew == lColor );
        lColor = lNew;
    }

    WinReleasePS( hPS );
}


/*  Behandelt Scrollbalken-Meldungen von der Steuerungs-
 *  Dialogbox. Korrigiert die Scrollbalkenposition, wobei die
 *  Grenzen berücksichtigt werden, kopiert den Scrollbalkenwert
 *  in das daneben liegende Editierfeld und stellt dann
 *  nDistance oder nInterval entsprechend ein.
 */

VOID SayDoBarMsg( hWndDlg, idBar, wCode, nThumb )
    HWND        hWndDlg;
    USHORT      idBar;
    USHORT      wCode;
    SHORT       nThumb;
{
    SHORT       nPos;
    SHORT       nOldPos;
    ULONG       lRange;

    nOldPos = nPos =
        (SHORT)WinSendDlgItemMsg(
            hWndDlg, idBar, SBM_QUERYPOS, 0L, 0L
        );

    lRange =
        WinSendDlgItemMsg(
            hWndDlg, idBar, SBM_QUERYRANGE, 0L, 0L
        );

    switch( wCode )
    {
        case SB_LINEUP:         --nPos;                     break;

        case SB_LINEDOWN:       ++nPos;                     break;

        case SB_PAGEUP:         nPos -= 10;                 break;

        case SB_PAGEDOWN:       nPos += 10;                 break;

        case SB_THUMBPOSITION:
        case SB_THUMBTRACK:     nPos = nThumb;              break;

        case SB_TOP:            nPos = LOUSHORT(lRange);    break;

        case SB_BOTTOM:         nPos = HIUSHORT(lRange);    break;
    }
```

```
    if( nPos < nMin )
        nPos = nMin;

    if( nPos > nMax )
        nPos = nMax;

    if( nPos == nOldPos )
        return;

    SetScrollPos( hWndBar, SB_CTL, nPos, TRUE );

    SetDlgItemInt( hWndDlg, idBar+1, nPos, FALSE );



    switch( idBar )
    {
      case ITEM_DISTBAR:
        nDistance = nPos;
        break;

      case ITEM_INTBAR:
        KillTimer( hWndWhat, TIMER_MOVE );
        nInterval = nPos;
        SetTimer( hWndWhat, TIMER_MOVE, nInterval, NULL );
        InvalidateRect( hWndWhat, NULL, FALSE );
        break;
    }
}



/*  Beendet die Anwendung und gibt alle verwendeten Ressourcen
 *  frei. Diese Funktion kehrt NICHT zum Aufrufer zurück, das
 *  Progamm wird beendet.
 */

void SayExitApp( nRet )
    int         nRet;
{
    if( GetModuleUsage(hInstance) == 1 ) {
        DeleteObject( hbrBkgd );
    }

    exit( nRet );
}



/*  Füllt ein angegebenes Rechteck mit der Hintergrundfarbe.
 *  Prüft vorher, daß das Rechteck nicht leer ist.
 */

void SayFillRect( hDC, nLeft, nTop, nRight, nBottom )
    HDC         hDC;
    int         nLeft;
    int         nTop;
    int         nRight;
    int         nBottom;

{
    RECT        rcFill;

    if( nLeft >= nRight  ||  nTop >= nBottom )
        return;


    SetRect( &rcFill, nLeft, nTop, nRight, nBottom );

    FillRect( hDC, &rcFill, hbrBkgd );
}
```

```
    if( nPos < LOUSHORT(lRange) )
        nPos = LOUSHORT(lRange);

    if( nPos > HIUSHORT(lRange) )
        nPos = HIUSHORT(lRange);

    if( nPos == nOldPos )
        return;

    WinSendDlgItemMsg(
        hWndDlg, idBar, SBM_SETPOS, (LONG)nPos, ØL
    );

    WinSetDlgItemInt( hWndDlg, idBar+1, nPos, FALSE );

    switch( idBar )
    {
      case ITEM_DISTBAR:
        nDistance = nPos;
        break;

      case ITEM_INTBAR:
        WinStopTimer( hWndWhat, TIMER_MOVE );
        nInterval = nPos;
        WinStartTimer( hAB, hWndWhat, TIMER_MOVE, nInterval );

        WinInvalidateRect( hWndWhat, NULL );
        break;
    }
}



/*  Beendet die Anwendung und gibt alle verwendeten Ressourcen
 *  frei. Diese Funktion kehrt NICHT zum Aufrufer zurück, das
 *  Progamm wird beendet.
 */

VOID SayExitApp( nRet )
    INT         nRet;
{
    if( hWndWhatFrame )
        WinDestroyWindow( hWndWhatFrame );

    if( hMsgQ )
        WinDestroyMsgQueue( hMsgQ );

    if( hAB )
        WinTerminate( hAB );

    exit( nRet );
}


/*  Füllt ein angegebenes Rechteck mit der Hintergrundfarbe.
 *  Prüft vorher, daß das Rechteck nicht leer ist.
 */

VOID SayFillRect( hPS, xLeft, xBottom, xRight, xTop )
    HPS         hPS;
    SHORT       xLeft;
    SHORT       xBottom;
    SHORT       xRight;
    SHORT       xTop;

{
    RECT        rcFill;

    if( xLeft >= xRight  ||  xBottom >= xTop )
        return;

    WinSetRect( hAB, &rcFill, xLeft, xBottom, xRight, xTop );

    WinFillRect(
        hPS,
        &rcFill,
        WinQuerySysColor( hAB, SCLR_WINDOW )
    );
}
```

```
/*  Initialisiert die Anwendung.  */

BOOL SayInitApp( hPrevInstance, nCmdShow )
    HANDLE      hPrevInstance;
    int         nCmdShow;
{
    WNDCLASS    Class;
    HDC         hDC;
    TEXTMETRIC  Metrics;

    LoadString(
        hInstance, STR_NAME,  szAppName, sizeof(szAppName)
    );

    LoadString(
        hInstance, STR_TITLE, szTitle,   sizeof(szTitle)
    );

    LoadString(
        hInstance, STR_WHAT,  szText,    sizeof(szText)
    );


    hDC = CreateIC( "DISPLAY", NULL, NULL, NULL );
    GetTextMetrics( hDC, &Metrics );
    nDisplayPlanes = GetDeviceCaps( hDC, PLANES );
    DeleteDC( hDC );

    ptCharSize.x = Metrics.tmMaxCharWidth;
    ptCharSize.y = Metrics.tmHeight;

    ptScreenSize.x = GetSystemMetrics(SM_CXSCREEN);
    ptScreenSize.y = GetSystemMetrics(SM_CYSCREEN);

    if( ! hPrevInstance ) {

        hbrBkgd = CreateSolidBrush(GetSysColor(COLOR_WINDOW));

        Class.style          = Ø; /* CS_HREDRAW | CS_VREDRAW; */
        Class.lpfnWndProc    = SayWhatWndProc;
        Class.cbClsExtra     = Ø;
        Class.cbWndExtra     = Ø;
        Class.hInstance      = hInstance;
        Class.hIcon          = NULL;
        Class.hCursor        = LoadCursor( NULL, IDC_ARROW );
        Class.hbrBackground  = COLOR_WINDOW + 1;
        Class.lpszMenuName   = szAppName;
        Class.lpszClassName  = szAppName;


        if( ! RegisterClass( &Class ) )
            return FALSE;

    } else {

        GetInstanceData(
            hPrevInstance, (NPSTR)&hbrBkgd, sizeof(hbrBkgd)
        );
    }

    hWndWhat = CreateWindow(
        szAppName,
        szTitle,
        WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, Ø,
        CW_USEDEFAULT, Ø,
        (HWND)NULL,
        (HMENU)NULL,
        hInstance,
        (LPSTR)NULL
    );

    if( ! hWndWhat )
        return FALSE;

    ShowWindow( hWndWhat, nCmdShow );
    UpdateWindow( hWndWhat );

    return TRUE;
}
```

```
/*  Initialisiert die Anwendung.  */

BOOL SayInitApp()
{
    HPS         hPS;
    BOOL        bOK;

    hAB = WinInitialize();
    if( ! hAB )
        return FALSE;

    hMsgQ = WinCreateMsgQueue( hAB, Ø );
    if( ! hMsgQ )
        return FALSE;

    WinLoadString(
        hAB, NULL, STR_NAME,  szAppName, sizeof(szAppName)
    );
    WinLoadString(
        hAB, NULL, STR_TITLE, szTitle,   sizeof(szTitle)
    );

    WinLoadString(
        hAB, NULL, STR_WHAT,  szText,    sizeof(szText)
    );

    bOK = WinRegisterClass(
        hAB,
        szAppName,
        (LPFNWP)SayWhatWndProc,
        CS_SYNCPAINT,
        Ø,
        NULL
    );
    if( ! bOK )
        return FALSE;


    hWndWhatFrame = WinCreateStdWindow(
        (HWND)NULL,
        FS_TITLEBAR | FS_SYSMENU |
            FS_MENU | FS_MINMAX  | FS_SIZEBORDER,
        szAppName,
        szTitle,
        ØL,
        (HMODULE)NULL,
        MENU_WHAT,
        &hWndWhat
    );

    if( ! hWndWhatFrame )
        return FALSE;

    WinShowWindow( hWndWhat, TRUE );


    return TRUE;
}
```

```
/*  Initialisiert einen Scrollbalken in der Dialogbox. */

void SayInitBar( hWndDlg, idBar, nValue, nMin, nMax )
    HWND        hWndDlg;
    int         idBar;
    int         nValue;
    int         nMin;
    int         nMax;
{
    HWND        hWndBar;

    hWndBar = GetDlgItem( hWndDlg, idBar );

    SetScrollRange( hWndBar, SB_CTL, nMin, nMax, FALSE );
    SetScrollPos( hWndBar, SB_CTL, nValue, FALSE );

    SetDlgItemInt( hWndDlg, idBar+1, nValue, FALSE );
}

/*  Erklärt den Text im Hauptfenster für ungültig und
 *  korrigiert das Textrechteck, wenn es außerhalb der
 *  Grenzen liegt.
 */

void SayInvalidateText( hWnd )
    HWND        hWnd;
{
    SayLimitTextPos( hWnd );
    InvalidateRect( hWnd, &rcText, FALSE );
}

/*  Prüft die Textposition gegen den Anwendungsbereich des
 *  Fensters. Wenn der Text in irgendeiner Richtung aus dem
 *  Fenster gerät, wird die Position so korrigiert, daß er
 *  wieder darin liegt. Ändert den Wert von ptAdvance, so daß
 *  der Text scheinbar vom Rand abprallt. Behandelt beide
 *  Fälle, Fenster und Sinnbild.
 */

void SayLimitTextPos( hWnd )
    HWND        hWnd;
{
    RECT        rcClient;
    POINT       ptTextSize;

    ptTextSize = ptCharSize;

    if( ! bIconic ) {
        pText = szText;
        ptTextSize.x *= strlen(szText);
    }

    GetClientRect( hWndWhat, &rcClient );

    if( rcText.left > rcClient.right - ptTextSize.x ) {
        rcText.left = rcClient.right - ptTextSize.x;
        ptAdvance.x = -ptAdvance.x;
    }
    if( rcText.left < rcClient.left ) {
        rcText.left = rcClient.left;
        ptAdvance.x = -ptAdvance.x;
    }
    if( rcText.top > rcClient.bottom - ptTextSize.y ) {
        rcText.top = rcClient.bottom - ptTextSize.y;
        ptAdvance.y = -ptAdvance.y;
    }
    if( rcText.top < rcClient.top ) {
        rcText.top = rcClient.top;
        ptAdvance.y = -ptAdvance.y;
    }

    rcText.right  = rcText.left + ptTextSize.x;
    rcText.bottom = rcText.top  + ptTextSize.y;
}
```

```
/*  Initialisiert einen Scrollbalken in der Dialogbox. */

VOID SayInitBar( hWndDlg, idBar, nValue, nMin, nMax )
    HWND        hWndDlg;
    SHORT       idBar;
    SHORT       nValue;
    SHORT       nMin;
    SHORT       nMax;
{
    HWND        hWndBar;

    hWndBar = WinWindowFromID( hWndDlg, idBar );

    WinSendDlgItemMsg(
        hWndDlg,
        idBar,
        SBM_SETSCROLLBAR,
        (LONG)nValue,
        MAKELONG( nMin, nMax )
    );
    WinSetDlgItemInt( hWndDlg, idBar+1, nValue, FALSE );
}

/*  Erklärt den Text im Hauptfenster für ungültig und
 *  korrigiert das Textrechteck, wenn es außerhalb der
 *  Grenzen liegt.
 */

VOID SayInvalidateText( hWnd )
    HWND        hWnd;
{
    SayLimitTextPos( hWnd );
    WinInvalidateRect( hWnd, &rcText );
}

/*  Prüft die Textposition gegen den Anwendungsbereich des
 *  Fensters. Wenn der Text in irgendeiner Richtung aus dem
 *  Fenster gerät, wird die Position so korrigiert, daß er
 *  wieder darin liegt. Ändert den Wert von ptAdvance, so daß
 *  der Text scheinbar vom Rand abprallt. Behandelt beide
 *  Fälle, Fenster und Sinnbild.
 */

VOID SayLimitTextPos( hWnd )
    HWND        hWnd;
{
    RECT        rcClient;
    POINT       ptTextSize;

    ptTextSize = ptCharSize;

    if( ! bIconic ) {
        npszText = szText;
        ptTextSize.x *= strlen(szText);
    }

    WinQueryWindowRect( hWndWhat, &rcClient );

    if( rcText.xLeft > rcClient.xRight - ptTextSize.x ) {
        rcText.xLeft = rcClient.xRight - ptTextSize.x;
        ptAdvance.x = -ptAdvance.x;
    }
    if( rcText.xLeft < rcClient.xLeft ) {
        rcText.xLeft = rcClient.xLeft;
        ptAdvance.x = -ptAdvance.x;
    }
    if( rcText.yBottom < rcClient.yBottom ) {
        rcText.yBottom = rcClient.yBottom;
        ptAdvance.y = -ptAdvance.y;
    }
    if( rcText.yBottom > rcClient.yTop - ptTextSize.y ) {
        rcText.yBottom = rcClient.yTop - ptTextSize.y;
        ptAdvance.y = -ptAdvance.y;
    }

    rcText.xRight = rcText.xLeft   + ptTextSize.x;
    rcText.yTop   = rcText.yBottom + ptTextSize.y;
}
```

```
/*  Bewegt den Text im Fenster indem die alte Position für
 *  ungültig erklärt wird, rcText entsprechend ptMove angepaßt
 *  wird und dann die neue Position ungültig gemacht wird.
 */

void SayMoveText( hWnd, ptMove )
    HWND        hWnd;
    POINT       ptMove;
{
    SayInvalidateText( hWnd );
    rcText.left = ptMove.x - (rcText.right -rcText.left >> 1);

    rcText.top  = ptMove.y - (rcText.bottom-rcText.top  >> 1);

    SayInvalidateText( hWnd );
}


/*  Stellt einen der Dialogscrollbalken auf *pnValue. Wenn der
 *  Wert außerhalb der Grenzen liegt wird er auf den gültigen
 *  Bereich begrenzt und *pnValue auch auf diesen Wert
 *  eingestellt.
 */

void SaySetBar( hWndDlg, pnValue, idBar )
    HWND        hWndDlg;
    int *       pnValue;
    int         idBar;
{
    HWND        hWndBar;
    int         nMin;
    int         nMax;
    int         nValue;
    BOOL        bOK;

    hWndBar = GetDlgItem( hWndDlg, idBar );

    GetScrollRange( hWndBar, SB_CTL, &nMin, &nMax );

    nValue = GetDlgItemInt( hWndDlg, idBar+1, &bOK, FALSE );

    if( bOK  &&  nValue >= nMin  &&  nValue <= nMax ) {
        *pnValue = nValue;
        SetScrollPos( hWndBar, SB_CTL, nValue, TRUE );
    } else {
        SetDlgItemInt(
            hWndDlg,
            idBar+1,
            GetScrollPos( hWndBar, SB_CTL ),
            FALSE
        );
    }
}


/*  Dialogfunktion für die Dialogbox.  */

BOOL FAR SayWhatDlgProc( hWndDlg, wMsg, wParam, lParam )
    HWND        hWndDlg;
    unsigned    wMsg;
    WORD        wParam;
    LONG        lParam;
{
    HWND        hWndBar;
    RECT        rcWin;
    int         n;

    switch( wMsg )
    {
      case WM_COMMAND:
        switch( wParam )
        {
```

```
/*  Bewegt den Text im Fenster indem die alte Position für
 *  ungültig erklärt wird, rcText entsprechend ptMove angepaßt
 *  wird und dann die neue Position ungültig gemacht wird.
 */

VOID SayMoveText( hWnd, ptMove )
    HWND        hWnd;
    POINT       ptMove;
{
    SayInvalidateText( hWnd );
    rcText.xLeft   =
        ptMove.x - (rcText.xRight - rcText.xLeft    >> 1);
    rcText.yBottom =
        ptMove.y - (rcText.yTop   - rcText.yBottom  >> 1);
    SayInvalidateText( hWnd );
}


/*  Stellt einen der Dialogscrollbalken auf *pnValue. Wenn der
 *  Wert außerhalb der Grenzen liegt wird er auf den gültigen
 *  Bereich begrenzt und *pnValue auch auf diesen Wert
 *  eingestellt.
 */

VOID SaySetBar( hWndDlg, pnValue, idBar )
    HWND        hWndDlg;
    SHORT *     pnValue;
    SHORT       idBar;
{
    ULONG       lRange;
    SHORT       nValue;
    BOOL        bOK;

    lRange =
        WinSendDlgItemMsg(
            hWndDlg, idBar, SBM_QUERYRANGE, ØL, ØL
        );

    nValue =
        WinQueryDlgItemInt( hWndDlg, idBar+1, &bOK, FALSE );

    if(
        bOK  &&
        nValue >= LOUSHORT(lRange)  &&
        nValue <= HIUSHORT(lRange)
    ) {
        *pnValue = nValue;
        WinSendDlgItemMsg(
            hWndDlg, idBar,SBM_SETPOS, (LONG)nValue, (LONG)TRUE
        );
    } else {
        WinSetDlgItemInt(
            hWndDlg,
            idBar + 1,
            (INT)WinSendDlgItemMsg(
                hWndDlg, idBar, SBM_QUERYPOS, ØL, ØL
            ),
            FALSE
        );
    }
}


/*  Dialogfunktion für die Dialogbox.  */

ULONG FAR PASCAL SayWhatDlgProc(hWndDlg,wMsg, lParam1,lParam2)
    HWND        hWndDlg;
    USHORT      wMsg;
    ULONG       lParam1;
    ULONG       lParam2;
{
    HWND        hWndBar;
    RECT        rcWin;
    SHORT       n;

    switch( wMsg )
    {
    case WM_COMMAND:
        switch( LOUSHORT(lParam1) )
        {
```

```
        case IDOK:
          KillTimer( hWndWhat, TIMER_MOVE );
          GetDlgItemText(
              hWndDlg, ITEM_WHAT, szText, sizeof(szText)
          );
          if( strlen(szText) == Ø )
              LoadString(
                  hInstance, STR_WHAT, szText, sizeof(szText)
              );
          pText = szText;
          SaySetBar( hWndDlg, &nInterval, ITEM_INTBAR );
          SaySetBar( hWndDlg, &nDistance, ITEM_DISTBAR );
          SetTimer( hWndWhat, TIMER_MOVE, nInterval, NULL );
          InvalidateRect( hWndWhat, NULL, FALSE );
          return TRUE;


        case IDCANCEL:
          DestroyWindow( hWndDlg );
          return TRUE;

        case ITEM_CLEAN:
        case ITEM_FLICKER:
          bCleanPaint =
              IsDlgButtonChecked( hWndDlg, ITEM_CLEAN );
          return TRUE;
      }
      return FALSE;

    case WM_HSCROLL:
      if( HIWORD(lParam) )
          SayDoBarMsg(
              hWndDlg, HIWORD(lParam), wParam, LOWORD(lParam)
          );
      return TRUE;

    case WM_INITDIALOG:
      GetWindowRect( hWndDlg, &rcWin );
      ClientToScreen( hWndWhat, (LPPOINT)&rcWin.left );
      ClientToScreen( hWndWhat, (LPPOINT)&rcWin.right );
      n = rcWin.right - ptScreenSize.x + ptCharSize.x;
      if( n > Ø )
          rcWin.left -= n;
      rcWin.left &= ~7;  /* auf Byte ausrichten */
      n = rcWin.bottom - ptScreenSize.y + ptCharSize.y;
      if( n > Ø )
          rcWin.top -= n;


      SetWindowPos(
          hWndDlg,
          (HWND)NULL,
          rcWin.left,
          rcWin.top,
          Ø, Ø,
          SWP_NOSIZE | SWP_NOZORDER |
            SWP_NOREDRAW | SWP_NOACTIVATE
      );
      SetDlgItemText( hWndDlg, ITEM_WHAT, szText );


      SendDlgItemMessage(
          hWndDlg, ITEM_WHAT,
          EM_LIMITTEXT, sizeof(szText)-1, ØL
      );

      SayInitBar(
          hWndDlg, ITEM_INTBAR,
          nInterval, MIN_INTERVAL, MAX_INTERVAL
      );
```

```
        case IDOK:
          WinStopTimer( hAB, hWndWhat, TIMER_MOVE );
          WinQueryWindowText(
              WinWindowFromID( hWndDlg, ITEM_WHAT ),
              szText,
              sizeof(szText)
          );

          if( strlen(szText) == Ø )
              WinLoadString(
                  hAB, NULL, STR_WHAT, szText, sizeof(szText)
              );

          npszText = szText;
          SaySetBar( hWndDlg, &nInterval, ITEM_INTBAR );
          SaySetBar( hWndDlg, &nDistance, ITEM_DISTBAR );
          WinStartTimer( hAB,hWndWhat,TIMER_MOVE, nInterval);
          WinInvalidateRect( hWndWhat, NULL );
          break;

        case IDCANCEL:
          WinDestroyWindow( hWndDlg );
          break;

        case ITEM_CLEAN:
        case ITEM_FLICKER:
          bCleanPaint = (BOOL)WinSendDlgItemMsg(
              hWndDlg, ITEM_CLEAN,
              BM_QUERYCHECK, ØL, ØL
          );
          break;
      }
      break;

    case WM_DESTROY:
      hWndPanel = NULL;
      break;

    case WM_HSCROLL:
      SayDoBarMsg(
          hWndDlg, LOUSHORT(lParam1),
          HIUSHORT(lParam2), LOUSHORT(lParam2)
      );
      break;

    case WM_INITDIALOG:
      WinQueryWindowRect( hWndDlg, &rcWin );
      WinMapWindowPoints(
          hWndWhat, (HWND)NULL, (LPPOINT)&rcWin.xLeft, 2
      );
      n = rcWin.xRight - ptScreenSize.x + ptCharSize.x;
      if( n > Ø )
          rcWin.xLeft -= n;
      rcWin.xLeft &= ~7;  /* auf Byte ausrichten */
      n = rcWin.yTop - ptScreenSize.y + ptCharSize.y;
      if( n > Ø )
          rcWin.yBottom -= n;

      WinSetWindowPos(
          hWndDlg,
          (HWND)NULL,
          rcWin.xLeft,
          rcWin.yBottom,
          Ø, Ø,
          SWP_MOVE
      );

      WinSetWindowText(
          WinWindowFromID( hWndDlg, ITEM_WHAT ), szText
      );

      WinSendDlgItemMsg(
          hWndDlg, ITEM_WHAT, EM_SETTEXTLIMIT,
          (LONG)sizeof(szText)-1, ØL
      );

      SayInitBar(
          hWndDlg, ITEM_INTBAR,  nInterval,
          MIN_INTERVAL, MAX_INTERVAL
      );
```

```
        SayInitBar( hWndDlg, ITEM_DISTBAR,
            nDistance, MIN_DISTANCE, MAX_DISTANCE );
        CheckDlgButton( hWndDlg, ITEM_CLEAN, TRUE );
        return TRUE;

      case WM_NCDESTROY:
        FreeProcInstance( lpfnDlgProc );
        hWndPanel = NULL;
        return FALSE;
    }
    return FALSE;
}

/*  Anzeigeprozedur für das Hauptfenster. Erledigt zur
 *  Demonstration sowohl die saubere als auch die flimmernde
 *  Anzeigemethode.
 */

void SayWhatPaint( hWnd )
    HWND        hWnd;
{
    PAINTSTRUCT ps;

    BeginPaint( hWnd, &ps );

    SetTextColor( ps.hdc, rgbTextColor );

    SayLimitTextPos( hWnd );

    if( bCleanPaint ) {

        /* Saubere Anzeige, redundantes Löschen vermeiden */

        TextOut(
            ps.hdc,
            rcText.left,
            rcText.top,
            pText,
            bIconic ? 1 : strlen(szText)
        );

        SayFillRect(
            ps.hdc,
            ps.rcPaint.left,
            ps.rcPaint.top,
            rcText.left,
            ps.rcPaint.bottom
        );

        SayFillRect(
            ps.hdc,
            rcText.left,
            ps.rcPaint.top,
            rcText.right,
            rcText.top
        );

        SayFillRect(
            ps.hdc,
            rcText.left,
            rcText.bottom,
            rcText.right,
            ps.rcPaint.bottom
        );

        SayFillRect(
            ps.hdc,
            rcText.right,
            ps.rcPaint.top,
            ps.rcPaint.right,
            ps.rcPaint.bottom
        );

    } else {
```

```
        SayInitBar(
            hWndDlg, ITEM_DISTBAR, nDistance,
            MIN_DISTANCE, MAX_DISTANCE
        );
        WinSendDlgItemMsg(
            hWndDlg, ITEM_CLEAN, BM_SETCHECK, (LONG)TRUE, 0L
        );
        break;
    }
    return WinDefDlgProc( hWndDlg, wMsg, lParam1, lParam2 );
}

/*  Anzeigeprozedur für das Hauptfenster. Erledigt zur
 *  Demonstration sowohl die saubere als auch die flimmernde
 *  Anzeigemethode.
 */

VOID SayWhatPaint( hWnd )
    HWND        hWnd;
{
    HPS         hPS;
    RECT        rcPaint;
    GPOINT      gptText;

    hPS = WinBeginPaint( hWnd, (HPS)NULL, &rcPaint );

    GpiSetColor( hPS, lColor );

    SayLimitTextPos( hWnd );

    gptText.x = (LONG)rcText.xLeft;
    gptText.y = (LONG)rcText.yBottom;

    if( bCleanPaint ) {

        /* Saubere Anzeige, redundantes Löschen vermeiden */

        GpiSetBackMix( hPS, MIX_OVERPAINT);

        GpiCharStringAt(
            hPS,
            &gptText,
            (LONG)( bIconic ? 1 : strlen(szText) ),
            npszText
        );

        SayFillRect(
            hPS,
            rcPaint.xLeft,
            rcPaint.yBottom,
            rcText.xLeft,
            rcPaint.yTop
        );

        SayFillRect(
            hPS,
            rcText.xLeft,
            rcText.yTop,
            rcText.xRight,
            rcPaint.yTop
        );

        SayFillRect(
            hPS,
            rcText.xLeft,
            rcPaint.yBottom,
            rcText.xRight,
            rcText.yBottom
        );

        SayFillRect(
            hPS,
            rcText.xRight,
            rcPaint.yBottom,
            rcPaint.xRight,
            rcPaint.yTop
        );
    } else {
```

```
        /* Flimmernde Anzeige, Hintergrund löschen und
           gewöhnlich anzeigen */

        FillRect( ps.hdc, &ps.rcPaint, hbrBkgd );

        TextOut(
            ps.hdc,
            rcText.left,
            rcText.top,
            pText,
            bIconic ? 1 : strlen(szText)
        );
    }

    EndPaint( hWnd, &ps );

    if( ! nInterval )
        SayAdvanceTextPos( hWnd );
}


/*  Fensterfunktion für das Hauptfenster.  */

LONG FAR SayWhatWndProc( hWnd, wMsg, wParam, lParam )
    HWND        hWnd;
    unsigned    wMsg;
    WORD        wParam;
    LONG        lParam;
{
    FARPROC     lpfnAbout;




    switch( wMsg )
    {
      case WM_COMMAND:
        switch( wParam )
        {
          case CMD_ABOUT:
            lpfnAbout =
                MakeProcInstance( SayAboutDlgProc, hInstance );
            DialogBox(
                hInstance, MAKEINTRESOURCE(DLG_ABOUT),
                hWnd, lpfnAbout
            );
            FreeProcInstance( lpfnAbout );
            return 0L;

          case CMD_EXIT:
            DestroyWindow( hWndWhat );
            return 0L;

          case CMD_WHAT:
            if( hWndPanel ) {
                BringWindowToTop( hWndPanel );
            } else {
                lpfnDlgProc =
                    MakeProcInstance(SayWhatDlgProc,hInstance);
                if( ! lpfnDlgProc )
                    return 0L;
                hWndPanel = CreateDialog(
                    hInstance,
                    MAKEINTRESOURCE(DLG_WHAT),
                    (HWND)NULL,
                    lpfnDlgProc
                );
                if( ! hWndPanel )
                    FreeProcInstance( lpfnDlgProc );
            }
        }
        break;

      case WM_CREATE:
        srand( (int)time(NULL) );
        SetTimer( hWnd, TIMER_MOVE, nInterval, NULL );
        return 0L;
```

```
        /* Flimmernde Anzeige, Hintergrund löschen und
           gewöhnlich anzeigen */

        WinFillRect(
            hPS,
            &rcPaint,
            WinQuerySysColor( hAB, SCLR_WINDOW )
        );

        GpiCharStringAt(
            hPS,
            &gptText,
            (LONG)( bIconic ? 1 : strlen(szText) ),
            npszText
        );
    }
    WinEndPaint( hPS );
    if( ! nInterval )
        SayInvalidateText( hWnd );
}


/*  Fensterfunktion für das Hauptfenster.  */

ULONG FAR PASCAL SayWhatWndProc( hWnd, wMsg, lParam1, lParam2 )
    HWND        hWnd;
    USHORT      wMsg;
    ULONG       lParam1;
    ULONG       lParam2;
{
    POINT       ptMouse;
    GSIZE       gsChar;
    HPS         hPS;
    LONG        ColorData[COLORDATAMAX];
    BOOL        bNowIconic;

    switch( wMsg )
    {
      case WM_BUTTON1DOWN:
        if( bMouseDown )
            break;
        WinStopTimer( hAB, hWnd, TIMER_MOVE );
        bMouseDown = TRUE;
        WinSetCapture( hAB, hWnd );
        ptMouse.x = LOUSHORT(lParam1);
        ptMouse.y = HIUSHORT(lParam1);
        SayMoveText( hWnd, ptMouse );
        return 0L;

      case WM_BUTTON1UP:
        if( ! bMouseDown )
            break;
        bMouseDown = FALSE;
        WinSetCapture( hAB, (HWND)NULL );
        ptMouse.x = LOUSHORT(lParam1);
        ptMouse.y = HIUSHORT(lParam1);
        SayMoveText( hWnd, ptMouse );
        WinStartTimer( hAB, hWnd, TIMER_MOVE, nInterval );
        return 0L;

      case WM_CHAR:
        if(
               ( LOUSHORT(lParam1) & KC_KEYUP )  ||
             ! ( LOUSHORT(lParam1) & KC_VIRTUALKEY )
        ) {
            break;
        }
        SayInvalidateText( hWnd );
        switch( HIUSHORT(lParam2) )
        {
          case VK_LEFT:
            rcText.xLeft -= ptCharSize.x;
            ptAdvance.x   = -1;
            ptAdvance.y   = 0;
            break;
          case VK_RIGHT:
            rcText.xLeft += ptCharSize.x;
            ptAdvance.x   = 1;
            ptAdvance.y   = 0;
            break;
```

```
case WM_DESTROY:
  if( hWndPanel )
      DestroyWindow( hWndPanel );
  PostQuitMessage( Ø );
  return ØL;

case WM_KEYDOWN:
  SayInvalidateText( hWnd );
  switch( wParam )
  {
    case VK_LEFT:
      rcText.left -= ptCharSize.x;
      ptAdvance.x  = -1;
      ptAdvance.y  = Ø;
      break;

    case VK_RIGHT:
      rcText.left += ptCharSize.x;
      ptAdvance.x  = 1;
      ptAdvance.y  = Ø;
      break;

    case VK_UP:
      rcText.top  -= ptCharSize.y >> 1;
      ptAdvance.x  = Ø;
      ptAdvance.y  = -1;
      break;

    case VK_DOWN:
      rcText.top  += ptCharSize.y >> 1;
      ptAdvance.x  = Ø;
      ptAdvance.y  = 1;
      break;

    default:
      return ØL;
  }

  SayInvalidateText( hWnd );
  nDistLeft = nDistance;
  return ØL;

case WM_LBUTTONDOWN:
  if( bMouseDown )
      break;

  KillTimer( hWnd, TIMER_MOVE );
  bMouseDown = TRUE;
  SetCapture( hWnd );
  SayMoveText( hWnd, MAKEPOINT(lParam) );
  break;

case WM_LBUTTONUP:
  if( ! bMouseDown )
      break;

  bMouseDown = FALSE;
  ReleaseCapture();
  SayMoveText( hWnd, MAKEPOINT(lParam) );
  SetTimer( hWnd, TIMER_MOVE, nInterval, NULL );
  break;

case WM_MOUSEMOVE:
  if( bMouseDown )
      SayMoveText( hWnd, MAKEPOINT(lParam) );
  break;

case WM_PAINT:
  SayWhatPaint( hWnd );
  return ØL;
```

```
    case VK_UP:
      rcText.yBottom -= ptCharSize.y >> 1;
      ptAdvance.x     = Ø;
      ptAdvance.y     = -1;
      break;
    case VK_DOWN:
      rcText.yBottom += ptCharSize.y >> 1;
      ptAdvance.x     = Ø;
      ptAdvance.y     = 1;
      break;

    default:
      return ØL;
  }

  SayInvalidateText( hWnd );
  nDistLeft = nDistance;
  return ØL;

case WM_COMMAND:
  switch( LOUSHORT(lParam1) )
  {
    case CMD_ABOUT:
      WinDlgBox(
          (HWND)NULL, hWnd, (LPFNWP)SayAboutDlgProc,
          NULL, DLG_ABOUT, NULL
      );
      return ØL;

    case CMD_EXIT:
      WinDestroyWindow( hWndWhatFrame );
      return ØL;

    case CMD_WHAT:
      if( hWndPanel ) {
          WinSetWindowPos(
              hWndPanel,
              HWND_TOP,
              Ø, Ø, Ø, Ø,
              SWP_ZORDER | SWP_ACTIVATE
          );
      } else {
          hWndPanel = WinLoadDlg(
              (HWND)NULL,
              (HWND)NULL,
              (LPFNWP)SayWhatDlgProc,
              NULL,
              DLG_WHAT,
              NULL
          );
      }
  }
  return ØL;

case WM_CREATE:
  /* Zeichen-/Bildschirmgröße und Anzahl Farben
     herausfinden */
  hPS = WinGetPS( hWnd );
  GpiQueryCharBox( hPS, &gsChar );
  GpiQueryColorData( hPS, (LONG)COLORDATAMAX, ColorData);
  WinReleasePS( hPS );
  lColorMax = ColorData[3];
  ptCharSize.x = gsChar.width;
  ptCharSize.y = gsChar.height;
  ptScreenSize.x =
    WinQuerySysValue( (HWND)NULL, SV_CXSCREEN );
  ptScreenSize.y =
    WinQuerySysValue( (HWND)NULL, SV_CYSCREEN );
  /* Timer initialisieren */
  srand( (INT)time(NULL) );
  WinStartTimer( hAB, hWnd, TIMER_MOVE, nInterval );
  return ØL;

case WM_DESTROY:
  if( hWndPanel )
     WinDestroyWindow( hWndPanel );
  WinPostQueueMsg( hMsgQ, WM_QUIT, ØL, ØL );
  return ØL;
```

```
    case WM_SIZE:
      if( wParam == SIZEICONIC ) {
          if( ! bIconic )
              SetTimer( hWnd, TIMER_CHAR, 1000, NULL );
          bIconic = TRUE;
      } else {
          if( bIconic )
              KillTimer( hWnd, TIMER_CHAR );
          bIconic = FALSE;
      }

      SayInvalidateText( hWnd );
      nDistLeft = 0;

      SayAdvanceTextPos( hWnd );
      return 0L;


    case WM_TIMER:
      switch( wParam )
      {

        case TIMER_MOVE:
          SayAdvanceTextPos( hWnd );
          break;

        case TIMER_CHAR:
          SayAdvanceTextChar( hWnd );
          break;
      }

      return 0L;
  }

  return DefWindowProc( hWnd, wMsg, wParam, lParam );
}


/*  Hauptfunktion der Anwendung.  */

void WinMain( hInst, hPrevInst, lpszCmdLine, nCmdShow )
    HANDLE      hInst;
    HANDLE      hPrevInst;
    LPSTR       lpszCmdLine;
    int         nCmdShow;
{
    MSG         msg;
    hInstance = hInst;

    if( ! SayInitApp( hPrevInst, nCmdShow ) )
        SayExitApp( 1 );

    while( GetMessage( &msg, NULL, 0, 0 ) ) {

        if( hWndPanel  &&  IsDialogMessage( hWndPanel, &msg ) )
            continue;

        TranslateMessage( &msg );
        DispatchMessage( &msg );
    }

    SayExitApp( msg.wParam );
}
```

Listing 6w: *Der Quellcode von SayWhat in* `SW.C`.

```
    case WM_ERASEBACKGROUND:
      return 1L;  /* nicht löschen */


    case WM_MINMAX:
      bNowIconic = ( LOUSHORT(lParam1) == SWP_MINIMIZE );
      if( bIconic != bNowIconic ) {
          bIconic = bNowIconic;
          if ( bIconic )
              WinStopTimer(  hAB, hWnd, TIMER_CHAR );
          else
              WinStartTimer( hAB, hWnd, TIMER_CHAR, 1000 );
      }
      return 1L;


    case WM_MOUSEMOVE:
      if( bMouseDown ) {
          ptMouse.x = LOUSHORT(lParam1);
          ptMosue.y = HIUSHORT(lParam1);
          SayMoveText( hWnd, ptMouse );
      }
      return 0L;


    case WM_PAINT:
      SayWhatPaint( hWnd );
      return 0L;


    case WM_SIZE:
      SayInvalidateText( hWnd);
      nDistLeft = 0;
      SayAdvanceTextPos( hWnd);
      return 0L;


    case WM_TIMER:
      switch( LOUSHORT(lParam1) ) {
          case TIMER_MOVE:
              SayAdvanceTextPos( hWnd );
              break;
          case TIMER_CHAR:
              SayAdvanceTextChar( hWnd );
              break;
      }
      return 0L;
  }
  return WinDefWindowProc( hWnd, wMsg, lParam1, lParam2 );
}


/*  Hauptfunktion der Anwendung.  */

void cdecl main( nArgs, pArgs )
    INT         nArgs;
    PSZ         pArgs;
{
    QMSG        qMsg;


    if( ! SayInitApp() )
        SayExitApp( 1 );


    while( WinGetMsg( hAB, &qMsg, (HWND)NULL, 0, 0 ) ) {

        if( hWndPanel  &&  WinProcessDlgMsg( hWndPanel, &qMsg))
            continue;

        WinDispatchMsg( hAB, &qMsg );
    }


    SayExitApp( 0 );
}
```

Listing 6pm: *Der Quellcode von SayWhat in* `SWP.C`.

Nach der Positionierung der Dialogbox initialisiert der WM_INITDIALOG-Code seine Steuerungsfenster. Interessant ist hier vor allem die Initialisierung des Scrollbalkens in SayInitBar. Die Windows-Version ruft zwei Scrollbalken-Funktionen auf, SetScrollRange und SetScrollPos, um den Scrollbalken zu initialisieren. Der Presentation Manager erledigt dies auf einmal mit der Meldung SBM_SETSCROLLBAR, die die Position und den Bereich einstellt. Sie werden auch überall beim Presentation Manager feststellen, daß dies mit Meldungen anstatt besonderer Funktionen erfolgt; viele Sonderfunktionen für Steuerungsfenster sind zugunsten von Meldungen beseitigt worden. Wenn Sie Funktionen vorziehen, können Sie sich immer noch entsprechende Funktionen, die diese Meldungen schicken, selbst schreiben.

Weitere interessante Meldungen in SayWhatDlgProc sind WM_COMMAND und WM_HSCROLL. Die letztere wird bei Aktivitäten in einem Scrollbalken der Dialogbox geschickt und sie ruft SayDoBarMsg auf, um die neue Position des Scrollbalkens und den entsprechenden Wert im Editierfeld einzustellen. Im Presentation Manager übergibt diese Meldung jedoch die ID des untergeordneten Fensters der Scrollbalkensteuerung anstelle der Fensterhandle. Dies ist sehr hilfreich, da die Fenster-ID benötigt wird, um festzustellen, mit welchem Scrollbalken man es zu tun hat.

Die Meldung WM_COMMAND wird geschickt, wenn der Benutzer einen Push- oder Radio-Button in der Dialogbox anklickt. (Sie wird auch geschickt, wenn in einem der Editierfelder etwas eingegeben wird, doch SayWhat ignoriert dies.) Die Bearbeitung von WM_COMMAND ist in beiden Versionen nahezu identisch, nur die Funktion IsDlgButtonChecked wird im Presentation Manager durch die Meldung BM_ QUERYCHECK ersetzt.

Zusammenfassung

Wie Sie sehen können, ist es nicht ganz trivial, eine Windows-Anwendung auf den Presentation Manager umzusetzen, doch es kann ziemlich mechanisch erfolgen. Die Gemeinsamkeiten beider Systeme sind größer als die Unterschiede. Wenn Sie die Windows-Programmierung beherrschen, haben Sie das komplizierteste bereits hinter sich, denn Sie wissen, wie stark sich eine Windows-Anwendung von einem konventionellen MS-DOS- oder OS/2-Programm unterscheidet. Wenn Sie bisher noch nicht mit der Windows-Programmierung begonnen haben, tja dann... Was höre ich da in der Ferne? Das sind doch Hochzeitsglocken! Kommen Sie nicht zu spät!

Michael Geary/jü

MS-OS/2 mit Presentation Manager

Hintergrundinformationen zu OS/2, Version 1.1

Die Version 1.1 von MS-OS/2 ist das zweite Release von Microsofts neuem Betriebssystem für IBM AT-kompatible und andere Rechner der IBM-Personal-System/2-Serie mit den Intel-Mikroprozessoren 80286 oder 80386. Wie schon die Version 1.0 ist Microsoft-OS/2, Version 1.1, ein Entwicklungsergebnis des »Joint Development Agreement« zwischen Microsoft und IBM.

Der wesentliche Unterschied zwischen der Version 1.0 und der Version 1.1 ist die grafische Bedienungsoberfläche. Sie ist jener Teil des Betriebssystems, den der Anwender auf dem Bildschirm sieht und mit dem er interaktiv arbeitet. Die Version 1.0 hat eine zeichenorientierte Bedienungsoberfläche, ähnlich derjenigen von MS-DOS, über die der Anwender die Befehle jeweils in eine Prompt-Kommandozeile eingibt. In der Version 1.1 wird die zeichenorientierte Bedienungsoberfläche durch eine fortschrittliche grafikorientierte Oberfläche ersetzt, die den Namen MS-OS/2-Presentation Manager hat.

In den folgenden Hintergrund-Informationen soll zunächst dargelegt werden, was der MS-OS/2-Presentation Manager ist und welche Vorteile er bietet. Das bezieht sich sowohl auf das Betriebssystem selbst, so wie es auch mit der Version 1.0 geliefert wird, als darüber hinaus auf die grafische Bedienungsoberfläche des Presentation Managers. Darauf folgend wird erläutert, in welchem Maße eine Kompatibilität zwischen den Anwendungsprogrammen besteht, die für MS-DOS, Microsoft Windows und MS-OS/2 geschrieben sind. Außerdem enthält dieser Beitrag Informationen darüber, in welcher Weise Microsoft den Software-Herstellern hilft, Anwendungen für das neue Betriebssystem zu entwickeln. Abschließend werden Systemvoraussetzungen und Verfügbarkeitsdaten des neuen Betriebssystems genannt.

Was ist der MS-OS/2-Presentation Manager?

Der MS-OS/2-Presentation Manager ist eine grafische Bedienungsoberfläche, die die zeichenorientierte Bedienungsoberfläche der Version 1.0 von MS-OS/2 ersetzt. Er ist eine Weiterentwicklung der grafischen Bedienungsoberfläche von Microsoft Windows, die im November 1985 erstmals vorgestellt wurde. Die Gestaltung des Presentation Managers gleicht derjenigen von Microsoft Windows 2.0 und Microsoft Windows/386, so daß Anwendern, die bereits mit diesen Bedienungsoberflächen Erfahrungen haben, der Umstieg leicht fällt. Anders als Microsoft Windows ist der MS-OS/2-Presentation Manager jedoch kein separates Produkt. Er ist vielmehr ein Bestandteil von MS-OS/2.

Wie Microsoft Windows 2.0, erlaubt der MS-OS/2-Presentation Manager die Teilung des Bildschirms in verschiedene Fenster, in denen unterschiedliche Anwendungen gleichzeitig ablaufen und der Anwender sehen kann, was sich gerade in den einzelnen Fenstern abspielt. So kann der Anwender z.B. eine Kalkulationstabelle in einem Fenster darstellen und die darauf basierende grafische Darstellung in einem anderen Fenster. Sobald der Anwender die Daten in der Kalkulationstabelle ändert, verändert sich auch die grafische Darstellung im anderen Fenster, so daß für den Anwender eine sofortige visuelle Rückkopplung der durchgeführten Änderung besteht. Ein anderes Beispiel: Der Anwender kann Datenbank-Abfragen in ein Fenster eingeben, während die Ergebnisse der vorherigen Abfragen noch in einem anderen Fenster angezeigt werden, so daß er nicht mehr erst warten muß, wenn er weiterarbeiten möchte.

Die Befehlseingabe beim Presentation Manager entspricht der von Microsoft Windows. Der Anwender wählt den gewünschten Befehl durch die Positionierung eines Bildschirmzeigers auf das betreffende Symbol oder er wählt auf die gleiche Weise das gewünschte Objekt aus einer Liste. Das Eintippen von Befehlen in eine Prompt-Kommandozeile entfällt also weitgehend. Sofern der Anwender aus einem Menü von Möglichkeiten auswählt, kann er Drop-Down-Menüs anklicken und für die Auswahl öffnen. Bei komplexeren Menüs erscheint eine Dialog-Box, die den Anwender auffordert, weitere Daten einzugeben oder eine zusätzliche Wahl zu treffen. Diese grafikorientierte Darstellung bietet dem Anwender sehr viel mehr Informationen als eine zeichenorientierte Bedienungsoberfläche. Darüber hinaus ist der Dialog mit dem Computer unmittelbarer und deshalb einfacher zu lernen.

Weitere Windows-ähnliche Eigenschaften des Presentation Managers bestehen darin, sowohl die Tastatur- wie auch die Maus-Eingabe zu unterstützen und die Möglichkeit, sowohl zeichenorientierte wie auch grafikorientierte Anwendungen ablaufen zu lassen. Zeichenorientierte Anwendungen werden allerdings auf dem Bildschirm auch nur in Zeichendarstellung wiedergegeben und erlauben nicht die Nutzung der grafischen Möglichkeiten des Presentation Managers.

Die Vorteile des MS-OS/2-Betriebssystems

MS-OS/2 ist das erste Betriebssystem für Personal-Computer, das speziell zur Nutzung der Möglichkeiten der 80286/80386-Mikroprozessoren entwickelt wurde. Um die Vorzüge von MS-OS/2 zu verstehen, ist es notwendig, zuerst ein wenig von der Arbeitsweise der Mikroprozessoren 80286 und 80386 zu wissen.

Die Mikroprozessoren 80286 und 80386 können in zwei verschiedenen Betriebsarten arbeiten, die als *Real Mode* und als *Protected Mode* bezeichnet werden. Die Auswahlmöglichkeit bei den Betriebsarten ist ein wichtiges Kennzeichen dieser beiden Prozessoren.

Wenn ein Computer auf 80286- oder 80386-Mikroprozessor-Basis unter MS-DOS läuft, so arbeitet der Mikroprozessor im Real Mode. In dieser Betriebsart arbeiten der 80286 und der 80386 so wie ihr Vorgänger, der Intel-8088-

Prozessor, der in IBM-PC, PC/XT und in dazu kompatiblen Computern zu finden ist. Der einzige Unterschied besteht darin, daß die leistungsfähigeren 80286 und 80386 die Daten schneller verarbeiten als der 8088. Darüber hinaus bieten diese beiden neueren Prozessoren im Real-Modus jedoch keine neuen Möglichkeiten. Deshalb können Anwendungsprogramme wie bisher auch nicht mehr als 640 Kbyte Hauptspeicher adressieren, so wie in MS-DOS. Der Anwender ist außerdem wie bisher darauf beschränkt, nur eine Anwendung zur selben Zeit zu bearbeiten.

Wenn ein Computer auf 80286- oder 80386-Prozessor-Basis jedoch unter MS-OS/2 läuft, arbeitet der Mikroprozessor im Protected-Modus. In dieser geschützten Betriebsart hat er Zugriff auf spezielle Möglichkeiten, die es beim 8088-Prozessor nicht gibt, z.B. auf die Fähigkeit, einen größeren Arbeitsspeicher als 640 KByte zu adressieren und verschiedene Anwendungen gleichzeitig zu verarbeiten. Eine Erläuterung dieser Fähigkeiten erfolgt im Anschluß.

Aufhebung der 640-Kbyte-Speicherbegrenzung

Unter MS-OS/2 können Anwendungsprogramme mit einem physikalischen Hauptspeicher bis 16 Mbyte (RAM) und mit einem virtuellen Speicher - bei dem die Festplatte als Quasi-Erweiterung des Arbeitsspeichers eingesetzt und die Daten nach Bedarf in den Hauptspeicher geladen werden - bis maximal 1 GByte arbeiten. Durch die Beseitigung der 640-Kbyte-Speicherbegrenzung dürfen Anwendungsprogramme nun viel größer sein als vorher. Für den Anwender bringt dies größere Funktionalität und einfachere Bedienung mit sich.

Multitasking-Fähigkeiten

MS-OS/2 hat hochentwickelte Multitasking-Fähigkeiten, einschließlich einer prioritätsgesteuerten Prozessor-Zuteilung. Dieses »Priority-based Scheduling« macht es möglich, bestimmten Anwendungen eine höhere Priorität zuzuordnen als anderen, so daß sie mehr »Prozessor-Arbeitszeit« zugeteilt bekommen und der Prozessor für sie häufiger zur Verfügung steht. Diese Art der Prozessor-Zuweisung steigert die Effizienz von Multitasking-Anwendungen erheblich.

Multitasking verbessert andererseits auch die Produktivität des Anwenders. So kann er z.B. ein Textdokument ausdrucken und bei der Druckerausgabe neue Daten in eine Kalkulationstabelle eingeben. Ein weiteres Beispiel: Der Anwender arbeitet an der Erstellung einer Zeichnung, während im Hintergrund das Anwendungsprogramm »Elektronische Post« läuft und eingehende Nachrichten sammelt.

Höhere Produktivität des Anwenders

Durch die neuen Möglichkeiten von MS-OS/2 werden Software-Entwickler in die Lage versetzt, eine neue Generation von Anwendungsprogrammen zu entwickeln, die sehr viel leistungsfähiger als bisherige Programme und darüber hinaus noch einfacher zu bedienen sind. Das Resultat ist ein erheblicher Gewinn an Anwender-Produktivität.

MS-OS/2 Version 1.1 bietet zusätzliche Vorteile

Mit dem zusätzlichen Presentation Manager und anderen neuen Erweiterungen des Betriebssystem-Kerns bietet MS-OS/2, Version 1.1, wesentliche Vorteile gegenüber der Version 1.0. Des besseren Überblicks wegen sind diese Vorteile im folgenden in zwei Gruppen gegliedert worden: Vorteile für den Anwender einerseits und Vorteile für den Software-Entwickler andererseits.

Vorteile für den Anwender

Folgende Vorteile hat der Anwender durch die Nutzung von MS-OS/2, Version 1.1:

1. Einfachere Bedienung.
2. Einheitlichkeit innerhalb des gesamten Betriebssystems.
3. Einheitlichkeit bei allen Anwendungen.
4. Datenaustausch zwischen Programmen auf Anwendungsebene.
5. Die Möglichkeit, größere Massenspeicher-Partitions zu benutzen.

Einfachere Handhabung

Die Anwender des MS-OS/2, Version 1.1, werden erfahren, daß eine grafische Bedienungsoberfläche einfacher zu erlernen und zu bedienen ist, als eine zeichenorientierte Bedienungsoberfläche. Menschen können mehr Informationen verarbeiten, die visuell aufgenommen wurden, als über irgendein anderes Sinnesorgan. Bei der grafischen Bedienungsoberfläche wird diese Tatsache als Vorteil genutzt. Durch die Symbole, »Drop-Down«-Menüs, Dialog-Boxen und andere grafische Möglichkeiten erklärt sich der MS-OS/2-Presentation Manger im wesentlichen selbst. Bei Benutzung einer Maus läuft der Lernvorgang bei MS-OS/2 noch unmittelbarer ab: Der Anwender zeigt einfach mit dem Bildschirmzeiger auf die gewünschte Funktion und drückt dann die Maustaste.

Einheitlichkeit bei verschiedenen Betriebssystemen

Ein anderer Vorteil der neuen Bedienungsoberfläche ist die Einheitlichkeit über verschiedene Betriebssysteme hinweg. Da aus der Sicht des Anwenders der MS-OS/2-Presentation Manager identisch zu Microsoft Windows 2.0 und Microsoft Windows/386 ist, gibt es nun eine einheitliche Bedienungsoberfläche für Computer auf MS-DOS- und auf MS-OS/2-Basis. Diese Einheitlichkeit erleichtert dem Anwender den Übergang von einem Betriebssystem zum anderen und minimiert die zum Erlernen des neuen Systems erforder-

liche Zeit. Außerdem wird Microsoft MS-OS/2 weiterentwickelt, um die Möglichkeiten des 80386-Prozessors voll zu nutzen: Somit wird es eine einheitliche Bedienungsoberfläche für drei Prozessor-Generationen geben, weil auch bei dieser Erweiterung der MS-OS/2-Presentation Manager erhalten bleibt.

Einheitlichkeit über Anwendungen hinweg

Ein Vorteil, den der Presentation Manager ebenfalls bietet, ist eine einheitliche Bedienungsoberfläche für verschiedene Anwendungen. Das MS-OS/2-Software-Entwicklungs-Kit von Microsoft enthält alle Informationen, die notwendig sind, damit Software-Entwickler Anwendungen entwickeln können, die genau dieselbe Bedienungsoberfläche haben, wie sie der Presentation Manager darstellt. Das Konzept einer einheitlichen Anwendungs-Bedienungsoberfläche verkürzt die Lernkurve des Anwenders und bewirkt, daß er in kürzerer Zeit produktiver wird. So wird der Anwender alsbald wissen - ohne in ein Handbuch schauen zu müssen - daß er immer das »File«-Menü anklicken muß, um eine Datei zu öffnen und das »System«-Menü, um eine Anwendung zu beenden. Diese Einheitlichkeit in der Menüstruktur engt dabei keineswegs die Kreativität der Software-Entwickler ein, die weiterhin die Freiheit haben, die Bildschirm-Darstellung und die Wahlmöglichkeiten für den Anwender in der Weise zu gestalten, wie sie dies möchten. Für den Bediener ist es jedoch viel einfacher, neue Anwendungsprogramme zu lernen, weil gleichartige Funktionen bei einer Vielzahl von Anwendungsprogrammen in derselben einheitlichen Weise gehandhabt werden.

Datenaustausch zwischen Programmen auf Anwendungsebene

Ein anderer Vorteil von MS-OS/2, Version 1.1, ist die verbesserte Kommunikation zwischen den Anwendungen. Bei Verwendung des »Electronic Clipboard« kann der Anwender z.B. ein Bild aus einer Grafikanwendung oder aus einer Kalkulationstabelle in ein Textdokument hineinkopieren. Darüber hinaus erlaubt die Funktion »Dynamischer Datenaustausch« die direkte Kommunikation von Anwendungen untereinander. Ein Beispiel hierfür: Der Anwender möchte ein Programm auf Tabellenkalkulations-Basis entwickeln, das die aktuellen Werte seiner Aktien anzeigt und dabei Kauf-/Verkaufsempfehlungen gibt. Das Programm könnte eine Kommunikations-Software nutzen, um in periodischen Intervallen die »Dow Jones«-Datenbank abzufragen und mit diesen Werten automatisch die Kalkulationstabelle aktualisieren.

Die Verwaltung großer Festplatten-Partitions

Die Version 1.1 umfaßt ein erweitertes File-System, das es erlaubt, Festplatten in größere Partitions zu unterteilen. Statt der Begrenzung auf 32 Mbyte pro Partition, wie sie bei MS-DOS und MS-OS/2, Version 1.0, vorhanden ist, erlaubt die Version 1.1 dem Anwender die Definition von Partitions bis zu einer Größe von 512 MByte - in Abhängigkeit davon, wie er seine Hardware konfiguriert.

Vorteile für den Software-Entwickler

Einer der Hauptvorteile des Presentation Managers für Software-Entwickler ist, daß zahlreiche Funktionen, die sie üblicherweise in die Anwendungsprogramme einbauen, bereits in dieser grafischen Bedienungsoberfläche vorhanden sind. Diese Funktionen umfassen unter anderem:

1. Geräteunabhängige grafische Anwendungsumgebung:
   Der Presentation Manager ist funktionskompatibel zu einer breiten Palette von Hardware-Einheiten und befreit damit den Software-Entwickler von dem Problem, sich um die Hardware-Kompatibilität sorgen zu müssen.
2. Hochentwickelte Grafik-Bibliothek:
   Software-Entwickler können nach Bedarf Funktionen aus der Bibliothek benutzen, so daß es einfacher für sie ist, grafisch orientierte Anwendungen zu entwickeln. Es ist z.B. nicht notwendig, daß der Software-Entwickler Programme für Dialog-Boxen, grafikspezifische Figuren wie Kreise, Rechtecke, etc. oder die Farbauswahl schreibt. Die erforderlichen Software-Routinen sind bereits in der Bibliothek vorhanden.
3. Eingebaute Fenstertechnik:
   Software-Entwickler werden es als relativ einfach empfinden, fensterorientierte Anwendungen zu entwickeln, weil die Fenster-Funktionen bereits im grafischen Bedienungs-Interface vorhanden sind.
4. Eingebaute Interprozeß-Kommunikation:
   Es wird in Zukunft sehr viel einfacher für Software-Entwickler sein, Anwendungen zu entwickeln, die Daten mit anderen Anwendungen austauschen, weil der Presentation Manager ein integriertes Programm für die Interprozeß-Anwendungs-Kommunikation enthält.

Weil alle diese Funktionen bereits im Presentation Manager vorhanden sind, werden Software-Entwickler weniger Programmcode selber schreiben müssen und deshalb in der Lage sein, ihr Produkt früher fertigzustellen. Ein anderer Vorteil für die Software-Entwickler liegt in dem stark wachsenden Markt für Anwendungsprogramme mit einer grafischen Bedienungsoberfläche und Fensterfunktionen. Weil die MS-OS/2-Version 1.1 das Standard-Bedienungsinterface für alle Computer auf OS/2-Basis sein wird, haben alle Anwendungen, die für den Presentation Manager geschrieben werden, einen großen potentiellen Markt. Aufgrund der Gleichartigkeit der Architektur des Presentation Managers mit derjenigen von Microsoft Windows 2.0 und Windows/386 wird es für die Anwendungsentwickler darüber hinaus einfach sein, Programme zu entwickeln, die sowohl unter MS-OS/2 als auch in einer MS-DOS-Umge-

Inserentenverzeichnis

bung unter Microsoft Windows laufen. Auch dies bedeutet eine Vergrößerung des Marktpotentials.

Anwendungskompatibilität in verschiedenen Betriebssystem-Umgebungen

Sowohl Software-Entwickler als auch Anwender werden sich fragen, in welchem Maße Anwendungsprogramme, die bereits für die drei PC-Betriebssystem-Umgebungen MS-DOS, Microsoft Windows und MS-OS/2 existieren, von einer Betriebssystem-Umgebung auf die andere übertragbar sind. Die Antwort hängt sowohl davon ab, in welcher Richtung die Portierung erfolgen soll - z.B. von MS-DOS zu MS-OS/2 oder umgekehrt - und andererseits, welcher Grad von Kompatibilität gefordert wird. Es gilt abzuwägen, ob die Anwendung in jeder Umgebung gleichermaßen laufen soll, oder ob sie die spezifischen Möglichkeiten der höherentwickelten Betriebssystem-Umgebungen nutzt.

Von MS-DOS zu OS/2: Bei der Übertragung von Anwendungsprogrammen von MS-DOS zu MS-OS/2 kann der Anwender davon ausgehen, daß die meisten seiner MS-DOS-Anwendungen auch unter der neuen Betriebssystem-Umgebung laufen; allerdings, jeweils nur eine Anwendung zur selben Zeit. Es ist nicht möglich, eine MS-DOS-Anwendung in mehreren Fenstern laufen zu lassen oder die Vorteile der speziellen MS-OS/2-Funktionen zu nutzen. Um dies zu tun, müssen die Software-Entwickler bereits vorhandene MS-DOS-Anwendungen modifizieren.

MS-Windows-Anwendungen auf MS-OS/2 übertragen: Anwendungsprogramme, die für die Windows-Umgebung geschrieben wurden, sind leichter auf MS-OS/2 zu übertragen, weil es hierbei das gemeinsame grafische Interface gibt. Natürlich sind trotzdem einige Änderungen notwendig, weil Windows ursprünglich für MS-DOS und nicht für MS-OS/2 entwickelt wurde.

Änderungen sind in folgenden Bereichen notwendig:

1. Im »Application Programming Interface«, um eine Anpassung an die Funktionen des Protected Mode von MS-OS/2 zu erreichen.
2. Änderungen bei den Aufrufen für die Grafik-Bibliothek, die bei MS-OS/2 erheblich verbessert wurde.
3. Änderungen in einigen Kodierungs-Regeln, die modifiziert wurden, um eine bessere Standardisierung im Hinblick auf MS-OS/2 zu erreichen.

Von MS-OS/2, Version 1.0, zu MS-OS/2, Version 1.1

Alle Anwendungen, die für die Version 1.0 von MS-OS/2 geschrieben wurden, werden auch ohne jegliche Änderungen unter der Version 1.1 laufen, weil Microsoft für die entsprechende Aufwärtskompatibilität gesorgt hat. Sofern der Entwickler seine Version-1.0-Anwendung gemäß den Richtlinien von Microsoft entwickelt hat, wird sie auch als eine fensterorientierte Anwendung unter Version 1.1 laufen.

Von MS-OS/2 zu MS-DOS

MS-OS/2-Anwendungen, die spezielle MS-OS/2-Funktionen nutzen, wie beispielsweise den vergrößerten Hauptspeicher oder das Multitasking, laufen nicht auf MS-DOS-Computern, weil diese Funktionen nur unter MS-OS/2 vorhanden sind.

Unterstützung für die Software-Entwickler

Um Software-Entwicklern beim Start der Entwicklung von MS-OS/2-Anwendungsprogrammen zu helfen, hat Microsoft ein MS-OS/2-Software-Development-Kit zusammengestellt. Dieses Kit enthält alles, was der Entwickler braucht, um mit der Entwicklungsarbeit für MS-OS/2-Anwendungen beginnen zu können.

Die derzeitige Version des Entwicklungskits beinhaltet die endgültige Version von MS-OS/2 1.0, technische Spezifikationen des MS-OS/2-Presentation Managers und des MS-OS/2-LAN-Managers, eine neue Version des Microsoft Macro-Assemblers und des C-Compilers sowie andere Software-Entwicklungs-Werkzeuge. Eine aktualisierte Version des Kits, die eine Vorversion des Presentation Managers enthält, ist ab Frühjahr 1988 lieferbar. Diese Version wird kostenlos an solche Entwickler abgegeben, die bereits das ursprüngliche Entwicklungs-Kit erworben haben. Dieser Update-Service ist Bestandteil des Maßnahmen-Katalogs, mit dem Microsoft MS-OS/2-Systementwickler unterstützt. Zu diesem Katalog gehören ferner die Dienstleistungen des Microsoft Instituts, MS-OS/2-Symposien und -Seminare, das Microsoft Dial (Online Informationsdatenbank) sowie das Ihnen vorliegende Microsoft System-Journal.

Hardware-Voraussetzungen

Die von Microsoft empfohlene Mindestkonfiguration zum Betrieb des MS-OS/2 ist ein Computer mit 2 MByte Hauptspeicher (RAM) und einer Festplatte. Soll der Presentation Manager eingesetzt werden, muß das System darüber hinaus einen Grafik-Bildschirm mit entsprechendem Grafikadapter haben.

Produkt-Verfügbarkeit

Wie MS-DOS wird MS-OS/2 nur an PC-Hersteller geliefert (OEM-Kunden). Die Hersteller adaptieren, testen und vermarkten das Betriebssystem auf ihren jeweiligen Computern. Die OEM-Lizenzkunden von Microsoft werden im vierten Quartal 1988 die vollständige Implementierung der MS-OS/2, Version 1.1, erhalten. Obwohl die Verfügbarkeit des Betriebssystems für die Endanwender von den Computerherstellern abhängt, die MS-OS/2-Lizenzen bei Microsoft erworben haben, geht Microsoft davon aus, daß MS-OS/2, Version 1.1, noch vor dem Jahresende 1988 für die Endanwender verfügbar ist.

Mitteilungen .... Mitteilungen .... Mitteilungen

Windows-Software-Development-Kit in der Version 2.0 verfügbar

Seit April liefert Microsoft die Version 2.0 des Windows-Software-Development-Kits aus. Das Kit wurde zur schnelleren Entwicklung von Anwendungen auf Windows-Basis ausgelegt. Das betrifft sowohl die Windows-Version 2.0 als auch die Windows-386-Version für die MS-DOS-Betriebssystem-Umgebung. Die Version 2.0 des Microsoft Windows-Software-Entwicklungs-Kits beinhaltet eine Vielzahl von Verbesserungen gegenüber früheren Versionen (Windows-SDK 1.0 und 1.04), einschließlich mehr als fünfzig neuer Funktionen und Meldungen; neue, erweiterte Utilities; eine vereinfachte Installation sowie eine umfangreichere Dokumentation. Das neue SDK-Paket ist außerdem mit 3½-Zoll-Disketten für die IBM-Personal System/2-Computer ausgestattet.

Alle bisherigen Windows-SDK-Utilities wurden in der Version 2.0 verbessert und eine Reihe neuer Utilities sind hinzugekommen. Der Leistungsumfang des neuen SDK-Pakets sieht folgendermaßen aus:

- Mehr als 50 neue Funktionen und Meldungen, die MS-OS/2-Programmfunktionen emulieren.
- Eine neue SNAP-Utility zur Definition und zum Ausdruck von Teilen des Bildschirms.
- Eine neue SPY-Utility zur Prüfung von Meldungen, die von Windows an die Anwendung gesendet werden.
- Eine neue Fließkomma-Mathematik-Bibliothek, die das Vorhandensein eines Fließkomma-Prozessors abfragt und ihn dann gegebenenfalls einsetzt.
- Einen verbesserten Dialog-Editor zur Erstellung von Dialog-Boxen und zum Sichern von Eingaben.
- Einen verbesserten ICON-Editor zur Erstellung spezieller Symbole, Cursor und Bit-Maps.
- Einen verbesserten Font-Editor zur Zeichengenerierung durch den Anwender.
- Einen verbesserten Ressource-Compiler zur Kompilierung von Anwendungs-Ressourcen-Files und zum Hinzufügen dieser Files zu den Hauptressourcen-Files.
- Eine verbesserte »Programm-Wartungs-Hilfe« zur automatischen Programmpflege.
- Einen verbesserten symbolischen Debugger.
- Eine neue Dokumentation, einschließlich eines Lernführers mit Quellcode-Beispielen für Windows-Programmieranfänger.
- Ein verbessertes Installationsprogramm mit anwenderfreundlichen Anleitungen.

MS-OS/2 LAN-Manager-Technik auf Unix-System-V portiert

Microsoft und Hewlett-Packard haben mit dem Microsoft LAN-Manager/X (LM/X) ein neues Netzwerk angekündigt, das die Funktionen des MS-OS/2 LAN-Managers in die Unix-System-V-Betriebssystemumgebung einbringt. Das neue Produkt ist gemeinsam von Microsoft und Hewlett-Packard entwickelt worden. Es ermöglicht Unix-System-V-basierenden Servern, auf denen LM/X installiert ist, Service-Anforderungen von Workstations zu unterstützen, die unter MS-Networks-, MS-Networks-for-Xenix und MS-OS/2 LAN-Manager-Netzwerksoftware laufen.

LM/X wird zuerst für Systeme mit dem Intel-80386-Prozessor entwickelt, die als Betriebssystem das aus Xenix/Unix-V abgeleitete Microsoft-Unix-System-V/386, Release 3.2, haben. Es kann jedoch auch auf Server portiert werden, die mit anderen Implementationen des Unix-Betriebssystems arbeiten. Microsoft wird LM/X-Lizenzen ab 1989 an OEM-Kunden vergeben. Ein Software-Entwicklungs-Kit von Microsoft soll gegen Ende 1988 lieferbar sein.

Die Netzwerk-Technik, ein entscheidender Teil heutiger Büroumgebung, macht es möglich, daß PCs über Gateways auf zentrale Ressourcen und Informationen Zugriff haben. Anwender von DOS- und OS/2-basierenden PCs können dank LM/X ohne zusätzliche Software mit einer breiten Palette von Computersystemen kommunizieren.

DCA und Microsoft kündigen strategische Zusammenarbeit an

Microsoft Corporation und die Digital Communications Associates Inc. (DCA) haben im März bekanntgegeben, daß sie im Rahmen einer strategischen Zusammenarbeit die MS-OS/2-Netzwerk-Fähigkeit weiterentwickeln. DCA hat eine Lizenz für den OS/2-LAN-Manager erworben, den das Unternehmen sowohl separat als auch in Verbindung mit dem DCA-Select-OS/2-Communications-Server (CS) vertreiben will. Microsoft bringt bei dieser Zusammenarbeit die technische Unterstützung zur Entwicklung des DCA-Select-OS/2-CS ein. Der DCA-Select-OS/2-CS bietet eine breite Palette von Kommunikationsmöglichkeiten, um lokale PC-Netzwerke mit Mainframe-Computern zu verbinden. Die Vermarktung erfolgt durch DCA.

Der Microsoft-OS/2-LAN-Manager läuft auf einer Reihe führender Netzwerke, einschließlich IBM-»Token Ring« und den weitverbreiteten Netzwerken auf Ethernet-Basis. Der DCA-Select-OS/2-CS setzt auf den Microsoft-OS/2-LAN-Manager auf. Die Microsoft-OS/2-LAN-Manager-Version von DCA wird die Bezeichnung DCA-Select-OS/2-LAN-Manager (LM) tragen und mit dem DCA-»10-Net« zusammenarbeiten. Dieses Produkt bietet damit einen MS-OS/2-Erweiterungspfad für mehr als 100.000 »10-Net«-Anwender.

Mitteilungen . . . . Mitteilungen . . . . Mitteilungen

Die Erweiterungsmöglichkeit stellt einen erheblichen Fortschritt im Bereich der LAN-Möglichkeiten dar. DCA verbindet damit seine führende Kommunikations-Hochleistungs-Software mit den fortschrittlichen Netzwerk-Fähigkeiten des Microsoft-OS/2-LAN-Managers, um auf einer ganz neuen Ebene von Funktionalität und Kosteneffizienz Mainframe-Anschlußmöglichkeiten für Arbeitsgruppen-Konfigurationen zu bieten.
Dadurch, daß der DCA-Select-OS/2-CS auf dem Microsoft-OS/2-LAN-Manager aufsetzt, kann DCA einen leistungsfähigen SNA-Kommunikationsserver bereitstellen, der sowohl MS-OS/2 wie auch MS-DOS-Workstations unterstützt. Der DCA-Select-OS/2-CS wurde gemäß der »Client/Server«-Architektur entwickelt, bei der kooperierende Elemente einer Applikation auf einem Server und einer Workstation lokalisiert sein können. Diese Methode schafft für DCA die Möglichkeit, einen Teil des Prozesses dem Server zuzuordnen, um damit Software-Pakete zu erstellen, die weniger Speicherbedarf in MS-DOS-Workstations benötigen. Darüber hinaus führt der Einsatz eines Servers zur Zusammenfassung und Übertragung aller Informationsanforderungen und damit zur Optimierung der Workstation-Nutzungszeit. Für den Zugriff auf den Mainframe durch die gesamte Netzwerk-Anwender-Basis ist nur eine Server-Verbindung erforderlich, woraus sich wiederum eine erhebliche Verminderung der Hardware-Kosten ergibt.

DCA entwickelt die Bediener-Oberfläche des DCA-Select-OS/2-CS in der Weise, daß sie konsistent zum »Common User Access Compliant Interface« des Microsoft-OS/2-LAN-Managers und zu MS-OS/2, Version 1.1, ist. Die Standard-IRMA-Bedieneroberfläche wird ebenfalls unterstützt.

»Auf MS-OS/2 basierende Server, die mit dem Microsoft-OS/2-LAN-Manager laufen, werden das Kernstück der zukünftigen PC-Netzwerke sein. Wir haben uns dafür entschieden, unseren Kommunikationsserver auf dem Microsoft-OS/2-LAN-Manager aufzubauen, weil wir davon ausgehen, daß dies der Standard werden wird. Durch das Angebot beider Produkte bieten wir eine komplette Lösung für solche Anwender, die einen Netzwerk-Zugriff über leistungsfähige SNA-Kommunikationseinrichtungen wünschen«, so kommentiert James Ottinger, Präsident von DCA, die Neuankündigung.

Als Teil der Zusammenarbeit beteiligte sich DCA an den »Microsoft Advanced Local Area Network Developers Conferences«, die vom 30. März bis zum 2. April in San Franzisko und vom 13. bis 15. April in New York stattfand. Eine vergleichbare europäische LAN-Konferenz plant Microsoft Ende Mai in Frankfurt durchzuführen.

Der Microsoft-OS/2-LAN-Manager erweitert die Leistung des Microsoft-OS/2-Betriebssystems zu der eines LAN-Servers. Er bietet eine Vielzahl von netzwerkorientierten Möglichkeiten, einschließlich transparentem File- und Drucker-Sharing, weitreichende Anwendersicherheit sowie Netzwerk-Administrations-Fähigkeiten. Er unterstützt darüber hinaus die Interprozeß-Kommunikation, so daß Entwickler eine neue Generation von dezentralen Applikationen gemäß der »Client/Server«-Architektur erstellen können. Der Microsoft-OS/2-LAN-Manager wird im zweiten Quartal 1988 für OEM-Kunden zur Verfügung stehen. Die Auslieferung des DCA-Select-OS/2-CS und des DCA-Select-OS/2-LM durch DCA ist für das erste Quartal 1989 geplant.

Vom Terminal zur Client/Server-Anwendung: »Extended Workgroup Computing«

Der oft als ein riesiger einheitlicher Markt betrachtete PC-Bereich besteht in Wirklichkeit aus einer Reihe von unterschiedlichen Anwendergruppen, z.B. Einzelanwender, Kleinunternehmen und Großanwender. Innerhalb der Großanwendergruppe lassen sich zwei Bereiche unterscheiden: Anwender, die von Terminal-/Mainframe-Konfigurationen zum PC kamen und solche, deren erste Erfahrungen mit dem Computer auf dem PC gemacht wurden. Bei der betrieblichen Datenverarbeitung der zuerst genannten Gruppe dominierte eine Konfiguration, in der 3270-Terminals an einen IBM-Mainframe angeschlossen waren. Im Jahre 1985 gab es beispielsweise in den Vereinigten Staaten nahezu 3,5 Millionen 3270-Terminals. Anwender, die zwischen einem »dummen« 3270-Terminal und einem PC »umschalten« wollen, benötigen eine Zusatzkarte, die den PC aus der Sicht des Mainframes zu einem 3270-Terminal macht. Karten dieser Art werden einfach in den PC eingesteckt, ohne daß eine Konfigurationsänderung beim Mainframe nötig ist oder daß der Anwender speziell hierfür ausgebildet wird. Die DCA-IRMA-Karte ist eine solche Zusatzkarte, die genau diese Funktion bietet. DCA verkaufte mehr als 600.000 IRMA-Karten und die IRMA-Karte wurde eines der am schnellsten verkauften neuen Hardware-Produkte in der Geschichte der Computer-Industrie. Nach der Vorstellung der IRMA-Karte durch DCA brachte auch IBM eine vergleichbare Karte auf den Markt. DCA erreichte mit der IRMA-Karte 55 Prozent Marktanteil im 3270-Einsteckkarten-Markt.

Als Auswirkung der Verbreitung von PC-basierenden Einsteckkarten wie der IRMA-Karte ging der Verkauf von 3270-Terminals seit 1987 zurück. Das Wachstum im 3270-Markt konzentrierte sich im wesentlichen auf PCs, die mit IRMA-Karten bzw. IRMA-ähnlichen Karten ausgerüstet sind. Personalcomputer, die 3270-Terminals emulieren, werden zur Zeit mit einer Stückzahl von rund 300.000 Systemen pro Jahr installiert, so daß hierauf zwischen 7 und 10 Prozent aller verkauften PCs entfallen.

Ein wesentlich größeres Segment, vermutlich mehr als 35 Prozent des gesamten PC-Marktes, bilden kommerzielle Anwender, die vor dem Erscheinen des PC keinen Zugriff auf irgendwelche Computer-Ressourcen hatten. Oftmals kauften Anwender, die in einer bestimmten Abteilung eines Unternehmens zusammenarbeiten, PCs, um bestimmte

Mitteilungen .... Mitteilungen .... Mitteilungen

Aufgaben damit zu erledigen, z.B. Tabellenkalkulations-Analysen oder Textverarbeitung. Nach ein paar Jahren erkannten diese Anwender die Notwendigkeit, Informationen gemeinsam mit solchen Leuten zu nutzen, mit denen sie regelmäßig zusammenarbeiteten. Das lokale Netzwerk bot hierfür die perfekte Lösung. Die erste bedeutende Generation von PC-Netzwerken, die 1984 auf der Basis des IBM-MS-NET-Standards entstand, bot den Anwendern einen Weg, gemeinsam auf Dateien zuzugreifen und teuere Peripheriegeräte, wie z.B. Laserdrucker, gemeinsam zu nutzen.

Der LAN-Markt entwickelte sich zunächst etwas langsamer als der »Mainframe Connectivity«-Markt. Obwohl MS-NET einen File-System-Standard auf hoher Ebene bietet, vermißte man einige Dinge darin, wie zum Beispiel Standards für die Hardware und die Verkabelung von PCs untereinander, ebenso wie Standards für die Kommunikationsprotokolle der unteren Ebenen. Dies steht in einem scharfen Kontrast zur Großcomputer-Umgebung, wo Produkte wie die IRMA-Karte von DCA einfach eingesteckt werden und zu existierenden IBM-Mainframe-Kommunikationsprotokollen kompatibel sind. Nach dem etwas langsamen Start legten die PC-LANs jedoch ständig zu. 1986 überstieg die Zahl der PC-Netzwerk-Karten bereits die Zahl der verkauften IRMA-3270-Karten. »Token Ring« und »Ethernet« wurden branchenweite Standards und erleichterten die Entscheidung der Kunden für ein PC-Netzwerk. Das Wachstum der PC-Netzwerke beschleunigte sich 1987. Es wurde mehr und mehr klar, daß sowohl aus technischen wie aus kaufmännischen Gründen die schnellere, leistungsfähigere und flexiblere LAN-Technik die dominierende Methode zur Verbindung von Computern und Peripheriegeräten wird, auch - in zunehmender Weise - bei Mainframe-Computern. Dieser Trend wurde bestätigt, als IBM 1986 ankündigte, daß seine Mainframe-Computer direkt an Token-Ring-Netzwerke anschließbar seien. 1987 begann IBM Produkte auszuliefern, z.B. das System 9370, die diese Netzwerk-Fähigkeiten beinhalten.

Trotz der rasch wachsenden Zahl von PC-LANs ist der 3270-Terminal-Markt keineswegs völlig verschwunden. PC-Netzwerke und direkter Mainframe-Anschluß waren zwei getrennte Welten. Diese Trennung in zwei Welten unterstützte den frühen und rasanten Einzug der PCs in kommerzielle Anwender-Umgebungen. Anwender mit spezifischen Anforderungen konnten durch die Installation von PCs ihre Probleme lösen, ohne dabei lange Planungszeiträume berücksichtigen zu müssen.

Die Kehrseite dieser Trennung bestand darin, daß ein PC-Anwender, der sowohl Zugriff auf die Ressourcen innerhalb einer Arbeitsgruppe als auch einen Mainframe haben wollte, zwei verschiedene Hardware-Karten in seinen PC einbauen mußte: je eine Netzwerk-Karte und Mainframe-Kommunikationskarte. Darüber hinaus bedurften beide Verfahren einer jeweils unabhängigen Administration. LAN-Gateway-Produkte wie DCA-IRMALAN übernahmen eine Pionierfunktion bei der Lösung dieses Problems. In reinen, auf DOS basierenden Umgebungen stießen diese Lösungen jedoch auf Grenzen, gezogen durch den Hauptspeicher und die Verarbeitungsleistung.

Ohne ein Multitasking-Betriebssystem mit Protected Mode wäre es nicht möglich, eine sichere Lösung zu implementieren, die den Zugriff auf ein lokales Netz und den Host-Anschluß bietet. MS-OS/2 ist das Kern-Betriebssystem, das diese Fähigkeit hat. Server, die direkt auf MS-OS/2 basieren wie der MS-OS/2-LAN-Manager und der IBM-OS/2-LAN-Manager stellen den nächsten Schritt dar, mit PCs ausgerüsteten Arbeitsgruppen sowohl Zugang zum Host als auch den Zugriff auf das lokale Netz zu bieten. Die Verfügbarkeit von Servern auf MS-OS/2-Basis wird einen erheblichen positiven Einfluß auf die Vielfalt von PC-Anwendungen unabhängiger Software-Hersteller haben. Die Entwickler können nun Anwendungen schreiben, die auf der »Client/Server«-Architektur basieren. In solchen Anwendungen laufen Teile der Software auf dem Anwender-PC: typischerweise die Software, mit der es der Anwender direkt zu tun hat, z.B. eine Tabellenkalkulation. Der andere Teil der Software läuft auf dem Server: typischerweise derjenige Teil, der die gemeinsamen System-Ressourcen steuert oder die externen Kommunikationskanäle verwaltet. Der DCA-Select-OS/2-CS ist ein hervorragendes Beispiel einer auf MS-OS/2-»Client/Server«-Anwendung. Produkte wie der DCA-Select-OS/2-CS werden eine Schlüsselrolle spielen, wenn es darum geht, das Konzept der »Extended Workgroup« Wirklichkeit werden zu lassen.

Multiplan 4.0 läuft unter MS-OS/2 und MS-DOS

Die Version 4.0 von MULTIPLAN wird nicht nur unter MS-DOS, sondern auch unter MS-OS/2, Version 1.0, laufen. Das Produkt, das in der englischen Version voraussichtlich im Sommer 1988 auf den Markt kommen wird, ist das erste einer Reihe von Microsoft-Anwendungspaketen, das die Möglichkeiten des neuen MS-OS/2-Betriebssystems nutzt. MULTIPLAN 4.0 verbindet die Leistungsfähigkeit und Flexibilität von MS-OS/2, Version 1.0, mit neuen Multiplan-Funktionen. Microsoft hat MULTIPLAN vollständig überarbeitet, um die Vorteile von MS-OS/2 wie Multitasking und den vergrößerten Arbeitsspeicher zu nutzen. MULTIPLAN 4.0 läuft als Protected Mode-MS-OS/2-Anwendung. Es ermöglicht echtes Multitasking, so daß der Anwender mehrere separate Kalkulationsdateien aufrufen kann und mehrere Funktionen gleichzeitig ausführbar sind.

»Mit MULTIPLAN 4.0 für MS-OS/2, Version 1.0, bietet Microsoft den weltweit 1,5 Mio. MULTIPLAN-Anwendern eine einfache Möglichkeit, schnell auf das MS-OS/2-Betriebssystem umzusteigen«, kommentiert Christian Wedell, Geschäftsführer der deutschen Microsoft GmbH, die Ankündigung. Gerade Tabellenkalkulierer werden die Neuerungen des MS-OS/2 gut nutzen können. Für sie fällt damit die 640-KByte-Speichergrenze von MS-DOS.

Termine ... Termine ... Termine ... Termine

Das Microsoft OS/2 Einführungs-Seminar

In diesem zweitägigen Seminar für PC-Software-Entwickler lernen die Teilnehmer das Konzept von MS-OS/2 kennen und erhalten einen Überblick seine Fähigkeiten und Programmierschnittstellen. Während des Seminars haben die Teilnehmer die Möglichkeit, das Gelernte anhand von Übungsaufgaben für sich selbst zu überprüfen.

Ort	Datum	Veranstalter
Hamburg	02./03.05.	OBZ
	16./17.05.	Integrata
Düsseldorf	09./10.05.	OBZ
	18./19.05.	OBZ
	26./27.05.	OBZ
	06./07.06.	OBZ
	22./23.06.	OBZ
	29./30.06.	OBZ
Köln	04./05.05.	OBZ
	16./17.05.	OBZ
	24./25.05.	OBZ
	30./31.05.	OBZ
	20./21.06.	OBZ
	27./28.06.	OBZ
Frankfurt	13./14.06.	OBZ
	20./21.06.	Integrata
Tübingen	09./10.05.	Integrata
	26./27.05.	Integrata
	27./28.06.	Integrata
München	02./03.05.	Integrata
	30./31.05.	Integrata
	15./16.06.	OBZ
Münster	06./07.06.	Integrata
Graz	26./27.05.	Ueberreuter
Wien	02./03.05.	Ueberreuter
	06./07.06.	Ueberreuter

Der Microsoft OS/2 Workshop

In diesem dreitägigen Seminar für PC-Software-Entwickler, die das MS-OS/2-Einführungsseminar besucht haben, lernen die Teilnehmer Family-API-Programme zu schreiben und Device-I/O-Routinen zu erstellen sowie Multitasking-Funktionen zu nutzen und eigene Dynamic-Link-Bibliotheken zu erstellen, sowie die erweiterten Speicherverwaltungsmöglichkeiten von OS/2 zu programmieren.

Ort	Datum	Veranstalter
Hamburg	04./05./06.05.	OBZ
Düsseldorf	08./09./10.06.	OBZ
Köln	18./19./20.05.	OBZ
	22./23./24.06.	OBZ
Münster	08./09./10.06.	Integrata
Frankfurt	22./23./24.06.	Integrata
Tübingen	13./14./15.06.	Integrata
München	04./05./06.05.	Integrata
Wien	04./05./06.05.	Ueberreuter
	08./09./10.06.	Ueberreuter

Das Microsoft Windows Einführungs-Seminar

In diesem zweitägigen Seminar für PC-Software-Entwickler lernen die Teilnehmer das Konzept von Microsoft Windows kennen und erhalten einen Überblick über dessen Fähigkeiten und Programmierschnittstellen.

Ort	Datum	Veranstalter
Hamburg	16./17.05.	OBZ
	06./07.06.	Integrata
Düsseldorf	02./03.05.	OBZ
	13./14.06.	OBZ
Köln	10./11.05.	OBZ
	06./07.06.	OBZ
Frankfurt	27./28.06.	Integrata
Tübingen	30./31.05.	Integrata
	20./21.06.	Integrata
München	16./17.05.	Integrata
	26./27.05.	Integrata
	30./31.05.	OBZ
	13./14.06.	Integrata
Wien	16./17.05.	Ueberreuter
	20./21.06.	Ueberreuter
Graz	06./07.06.	Ueberreuter
Linz	03./04.05.	Ueberreuter

Der Microsoft Windows Workshop

In diesem dreitägigen Seminar für PC-Software-Entwickler, die das Microsoft Windows Einführungsseminar besucht haben, lernen die Teilnehmer Benutzerschnittstellen zu erstellen, die grafische Programmierschnittstelle zu nutzen, die Routinen zum Memory Management anzuwenden und dynamische Bibliotheken zu erstellen und zu benutzen.

Ort	Datum	Veranstalter
Hamburg	18./19./20.05.	OBZ
	08./09./10.06.	Integrata
Düsseldorf	04./05./06.05.	OBZ
Köln	08./09./10.06.	OBZ
Münster	09./10./11.05.	Integrata
Tübingen	24./25./26.02.	Integrata
	04./05./06.05.	Integrata
München	02./03./04.03.	Integrata
	18./19./20.05.	Integrata
Wien	13./14./15.04.	Ueberreuter
	18./19./20.05.	Ueberreuter
	22./23./24.06.	Ueberreuter

Das Microsoft Institut arbeitet vor Ort mit kompetenten Schulungsunternehmen zusammen:

DPS Computer Schule GmbH, Braunschweig
Integrata GmbH, Tübingen
Olivetti Bildungs-Zentrum GmbH, Düsseldorf
Ueberreuter Media GmbH, Wien
Digicomp AG, Zürich
Electro Calcul SA, Lausanne.

Eine komplette Sprachenfamilie für MS-DOS und OS/2:

Neue Compiler für MS-OS/2-Applikationen

Microsoft hat im März fünf neue Sprachen, einen neuen »intelligenten« Texteditor für Programmierer, eine verbesserte Version des CodeView-Debuggers und andere Hilfen zur Entwicklung von Microsoft-OS/2- und IBM-OS/2-Applikationen angekündigt. Jeder Compiler wird in einer gemeinsamen Packung mit allen Disketten als MS-DOS- und gleichzeitig als MS-OS/2-Version ausgeliefert. Da sich die Preise der Compiler nicht erhöhen, kann der Umstieg von MS-DOS auf MS-OS/2 für die Benutzer der neuen Compiler-Versionen ohne Aufpreis erfolgen. Zusätzlich wird es die Möglichkeit zum Update von älteren MS-DOS-Compilern auf die neuen Versionen geben. Damit wird sowohl der Umstieg erleichtert als auch sichergestellt, daß es in kurzer Zeit viele weitere MS-OS/2-Anwendungen geben wird.

Die neuen Sprachversionen, der optimierende C-Compiler, Version 5.1, der Microsoft Macro Assembler, Version 5.1, die Microsoft Basic Compiler, Version 6.0, der optimierende Fortran Compiler, Version 4.1, und die neue Microsoft-Pascal-Compiler-Version 4.0 sind dazu geeignet, sowohl Anwendungen für den Real Mode von MS-DOS als auch für den Protected Mode von MS-OS/2 zu entwickeln. Ebenfalls angekündigt wurde das Microsoft-OS/2-»Programmers Toolkit«, das spezielle Dokumentationen und Hilfen zur Entwicklung von MS-OS/2-Anwendungen enthält.

Mit den neuen Produkten steht dem Entwickler eine komplette Palette von Entwicklungswerkzeugen zur Verfügung. »Unsere neuen Sprachversionen erlauben dem Entwickler die Auswahl von individuell gewünschten Sprachen«, kommentiert Christian Wedell, Geschäftsführer der deutschen Microsoft GmbH in München-Aschheim, die Ankündigung. Mit Hilfe der neuen Werkzeuge ist die Transformation »alter« Anwendungsprogramme in »neue« Anwendungen einfach.

Die Microsoft-Sprachenfamilie

Microsofts Sprachenfamilie hat einen hohen Grad an Gemeinsamkeiten. Alle fünf Microsoft-Sprachen beinhalten den Microsoft-Editor, einen rekonfigurierbaren Multifile-Multiwindow-Texteditor, der sowohl den Real Mode wie auch den Protected Mode unterstützt. CodeView, ebenfalls Bestandteil aller fünf Sprachen, unterstützt die Fehlersuche in MS-DOS- und MS-OS/2-Anwendungsprogrammen. Neue Möglichkeiten sind das »Blättern« in Daten (»Data Browsing«) und, für MS-OS/2, die Fehlersuche in Mehrfachthread-Anwendungsprogrammen und -Prozessen. Diese Art der Fehlersuche vereinfacht das Schreiben von komplexen MS-OS/2-Anwendungsprogrammen.

Alle Sprachen unterstützen den Aufruf von Routinen aus anderen Programmen (»Inter-Language Calling«). So kann beispielsweise eine Fortran-Routine aus einem C-Programm heraus aufgerufen werden. Das »Inter-Language Calling« vom Microsoft Macro Assembler aus wurde darüber hinaus vereinfacht. Alle fünf Sprachen sprechen das MS-OS/2-Betriebssystem direkt über das Anwendungs-Programmier-Interface (API) an. Sie arbeiten alle mit derselben mathematischen Bibliothek und unterstützen die Entwicklung sehr umfangreicher Programme - bis zu 16 Mbyte physikalischer Speicher und bis zu 1 GByte virtueller Speicher - in MS-OS/2-Umgebungen.

Weil alle fünf Sprachen sowohl unter MS-OS/2 als auch unter MS-DOS laufen und Programm-Code für beide Systeme erzeugen können, ist der Programmierer frei in der Wahl, welcher Umgebung er jeweils den Vorzug gibt. Er kann sein Zielprogramm so erstellen, daß es unter beiden Betriebssystemen läuft. Darüber hinaus können Microsofts C-, MASM-, Fortran- und Pascal-Produkte »FAPI«-Anwendungen erzeugen, in denen jeweils dieselbe ablauffähige Datei unverändert unter beiden Betriebssystemen laufen kann.

Glanzpunkte der Familie

Die folgenden fünf Hilfsprogramme sind Bestandteil aller fünf neuen Microsoft-Programmiersprachen-Pakete.

Microsoft CodeView-Debugger

Microsoft CodeView ist ein leistungsfähiger Debugger zur Fehlersuche und -behebung in MS-OS/2- und MS-DOS-Anwendungsprogrammen. Microsoft CodeView unterstützt alle fünf Microsoft-Sprachen und ermöglicht es dem Entwickler, frei zwischen der MS-OS/2- und der MS-DOS-Entwicklungsumgebung hin und her zu pendeln. Sowohl die Real Mode- wie auch die Protected Mode-Version bietet Textselektion am Bildschirm. Die neue Unterstützung für mehrere Threads in einem Prozeß unter MS-OS/2 beinhaltet: Einzelschritt-Schaltung auf Thread-Ebene, Ereignisverfolgung, bedingten Programmstop und Beobachtung des lokalen Stacks eines Threads. Zum ersten Mal kann der Programmierer Datenstrukturen sichtbar machen und verbundene Listen sowie verzweigte Strukturen verfolgen (»Data Browsing«). Microsoft CodeView ist in der Lage, die Fehlersuche in Dynamic Link Libraries und sehr großen Programmen - bis zu 128 Mbyte unter MS-OS/2 - zu unterstützen.

In ihrer Kombination vermindern diese neuen Möglichkeiten die Entwicklungszeit für komplexe Anwendungsprogramme drastisch und vereinfachen darüber hinaus den Aufwand, der erforderlich ist, um Anwendungsprogramme sowohl unter MS-OS/2 als auch unter MS-DOS zu warten.

Microsoft Editor

Der neue Microsoft Editor für MS-OS/2 und MS-DOS läßt dem Programmentwickler die freie Wahl bei der Entscheidung für eine Programmiersprache. Er ermöglicht das Öffnen von bis zu acht Fenstern und mehrerer Dateien gleichzeitig. Die Zahl der offenen Files hängt dabei vom verfügbaren Speicherplatz ab. Ein C-Programm läßt sich beispielsweise in einem Fenster editieren, ein Basic-Programm in einem anderen und ein Assembler-Programm in einem dritten Fenster. Der Editor ist »intelligent« genug, zu unterscheiden, welche Sprache in welchem Fenster aktiv ist, wenn der zugehörige Compiler aufgerufen wird.

Fehler des Compilers werden an den Editor zurückgemeldet und der Cursor in die Zeile plaziert, wo der Fehler auftrat. Der Microsoft Editor ist außerdem vollständig programmierbar, so daß Programmierer ihn den jeweiligen Bedürfnissen anpassen können. Die Tastatur läßt sich so konfigurieren, daß jeder beliebige Texteditor emuliert wird. Microsoft hat bereits Files zur Emulation von »Brief«, »Epsilon«, »Wordstar«, Microsoft QuickBasic und Microsoft QuickC, eingebaut. Unter MS-OS/2 nutzt der Editor die Vorteile des Multitasking, um z.B. im Hintergrund eine Kompilierung auszuführen, während der Programmierer eine neue Datei editiert.

Microsoft ILINK

Microsoft ILINK ist ein Inkremental-Linker ausschließlich für MS-OS/2. ILINK verbessert die Link-Geschwindigkeit um mehr als das zwanzigfache, weil er nur die ablauffähigen Module verbindet, die sich geändert haben.

Microsoft IMPLIB

Microsoft IMPLIB dient der Erzeugung von dynamischen Verbund-Bibliotheken unter MS-OS/2.

Microsoft BIND-Utility

Die Microsoft BIND-Utility ermöglicht die Erstellung von Anwendungsprogrammen, die sowohl unter DOS-3.X-Real-Mode als auch unter MS-OS/2-Protected-Mode laufen. BIND ist kein Bestandteil von Microsoft Basic 6.0.

Microsoft C Version 5.1 für MS-DOS und MS-OS/2

Microsoft C 5.1 unterstützt den professionellen Anwendungs-Programmierer in der Entwicklung, Kompilierung und Testphase sowohl unter dem bisherigen Betriebssystem MS-DOS, mit dem in dieser Betriebsart erreichbaren Real Mode der Prozessoren 8086, 8088, 80286 und 80386, als auch unter MS-OS/2, das erstmals den Protected Mode der Prozessoren 80286 und 80386 voll zu nutzen erlaubt. Weitreichende Systemunterstützung in Verbindung mit neuen Features der Code-Optimierung durch den Compiler, dem ebenfalls auf MS-OS/2 angepaßten Microsoft CodeView-Debugger, der Prototyp-Programmierung unter Microsoft QuickC und einer Reihe Utilities ergänzen das Programmiersprachen-Paket C 5.1.

Microsoft C 5.1 unterstützt die Betriebssysteme MS-DOS und MS-OS/2 in gleichem Maße. Die von den Prozessoren unter dem Betriebssystem MS-DOS bekannte Speichergrenze von 640 Kbyte kann erstmals durch Verwendung des Betriebssystems MS-OS/2 durchbrochen werden. Anwendungsprogramme dürfen somit einen bis zu 16 Mbyte umfangreichen RAM-Speicher verwenden und haben ferner die Möglichkeit, einen virtuellen Speicher der maximalen Größe von bis zu einem GByte zu nutzen. Microsoft C 5.1 unterstützt das Prinzip der Dynamic Link Libraries (DLL) und die Entwicklung von Multi-Thread-Anwendungen.

Ebenfalls im Sprachpaket enthalten sind ein Inkremental-Linker zur Verwendung in MS-OS/2, ein Editor, der im Protected Mode des Prozessors betrieben wird sowie eine Microsoft CodeView-Debugger-Version, die ebenfalls im Protected Mode des Prozessors betrieben wird und dort als komfortabler Debugger sowohl die Programm-Testphase als auch das Austesten der Dynamic Link Libraries gestattet. Microsoft C 5.1 enthält Microsoft QuickC, das Editierung, Compilierung, Debugging und Code-Ausführung innerhalb einer integrierten Umgebung für Real Mode-Anwendungen gestattet. Der Microsoft QuickC Editor erlaubt die schnelle Quellcode-Bearbeitung und akzeptiert sowohl das Tastatur-Interface von Microsoft Windows 2.0 als auch andere weitverbreitete Tastenkombinationen. Per »Hotkey« ist das Wechseln zwischen zwei verschiedenen Quelldateien möglich. Ferner werden unter Microsoft QuickC sowohl ein 25-Zeilen-Modus als auch ein 43-Zeilen-Modus auf einem EGA-Bildschirm unterstützt. Hervorstechendes Merkmal der QuickC-Programmierung ist die schnelle Umsetzung des Quellcodes in ablauffähigen Code und damit die problemlose Überprüfbarkeit und Änderungsmöglichkeit einer Routine. Die Fähigkeiten zur schnellen Überprüfung werden durch eine reduzierte Version des Microsoft CodeView-Debuggers unterstützt, der unter Microsoft QuickC als integrierter Debugger Einzelschritt-Debugging, fortlaufendes Debugging sowie Code-Ausführung bei voller Geschwindigkeit erlaubt. Zusätzlich unterstützt dieser Microsoft QuickC-Debugger die Überprüfung von Inhalten der Variablen oder mathematischen Ausdrücken während der Programmausführung.

Microsoft C 5.1 entspricht nicht nur vollkommen dem UNIX System V C, sondern zusätzlich auch den jüngsten Entwicklungen des ANSI-Standards. Als ANSI-Mitglied ist Microsoft maßgeblich an der Entwicklung und Realisierung eines möglichst breiten Standards beteiligt.

Um bestmögliche Flexibilität bei der Programmentwicklung zu erreichen, unterstützt Microsoft C 5.1 fünf verschiedene Speichermodelle, die unter den Begriffen *small*, *compact*, *medium*, *large* und *huge* bekannt sind. Darüber

hinaus führt Microsoft C auch für den Personalcomputer das »Mixed Model«-Konzept ein.

Kleine Produkt-Geschichte

Microsoft C wurde erstmals im Jahre 1983 eingeführt. Zwei Jahre später kündigte Microsoft die Version C 3.0 an. Dahinter verbarg sich ein weitgehend neues, intern erprobtes Produkt, das erstmals das Konzept des Mixed-Memory-Modells für den Personalcomputer anbot. Im Sommer 1986 brachte Microsoft die Compiler-Version 4.0 und dazu Microsoft CodeView, der Debugger-Fähigkeiten nun auch auf der Quellcode-Ebene erreichbar machte. Als erster optimierender Microsoft C-Compiler bedeutete die Version 4.0 einen entscheidenden Fortschritt und die Festlegung eines neuen Leistungsmaßstabs. Mit dem optimierenden Microsoft C-Compiler Version 5.0 im Jahr 1987 wurde ein weiterer Schritt in Richtung Verbesserung der Optimierungs-Technik gemacht. Zum selben Zeitpunkt wurde mit Microsoft QuickC 1.0 ein ideales Werkzeug sowohl zur Programmierung als auch zum Erlernen der Programmiersprache C angeboten. Außerdem wurde als Bestandteil des MS-OS/2 Software Development Kits mit Microsoft C 4.5 der erste Schritt zur Unterstützung des zukünftigen Betriebssystems getan. Microsoft C 5.1 verbindet die MS-OS/2 Unterstützung der Version 4.5 und die Optimierungs-Fortschritte des Microsoft C 5.0 in idealer Weise.

Microsoft Macro Assembler (MASM) 5.1 für MS-DOS und MS-OS/2

Der Microsoft Macro Assembler (MASM) 5.1 ist das aktuelle Angebot zur Entwicklung von Programmen in Assembler unter MS-DOS und MS-OS/2. Die Version 5.1 bietet ein leicht handhabbares Format und die Möglichkeit der Verbindung mit in Hochsprachen wie Microsoft Basic und QuickBasic, Microsoft C und QuickC, Fortran oder Pascal geschriebenen Routinen. In der aktuellen Version stehen alle Leistungen zur Programmentwicklung nunmehr unter den Betriebssystemen MS-DOS *und* MS-OS/2 zur Verfügung.

Der Microsoft Macro Assembler 5.1 wurde mit der Absicht entwickelt, ein möglichst einfach anwendbares Arbeitsmittel mit problemlosen Verbindungsmöglichkeiten zu in Hochsprachen geschriebenen Programmteilen zu schaffen. Spezielle Erweiterungen des MASM haben das Mixed-Language-Konzept, das die Verwendung aller Microsoft Sprachen in einem einzigen Programm erlaubt, auf so einfache Vorgänge wie das Erkennen der einzubindenden Sprache und der darin verwendeten auszutauschenden Parameter ausgedehnt. Zusätzlich sorgt eine Vielzahl an Beispielen in der überarbeiteten und erweiterten Dokumentation für hervorragenden Lehrstoff. Eine Hilfe besonderer Art leistet der CodeView-Debugger mit seiner Fähigkeit, Mixed-Language-Programme zu testen und automatisch der verwendeten Programmiersprache entsprechend zu identifizieren und den Quellcode dazu anzuzeigen.

Der Microsoft Macro Assembler unterstützt in seiner aktuellen Version den Befehlsvorrat des Intel Prozessors 80386, einschließlich der automatisierten Generierung des Codes sowohl für 16-Bit- als auch für 32-Bit-Segmente. Ferner wird automatisch die richtige Segmentierung für diesen Prozessortyp vorgenommen. Die Besonderheiten sowohl des Real Mode als auch des Protected Mode werden im gesamten Befehlsvorrat voll unterstützt. Anwendungen, die unter MS-OS/2 die 640-Kbyte-Grenze überschreiten, können ebenso entwickelt werden wie Dynamic Link Libraries und Multi-Threaded-Anwendungen erreichbar sind. Zusätzlich verfügt MASM 5.1 über die BIND-Utility, wodurch Programme in der Form einer .EXE-Datei erzeugt werden können, die sowohl unter dem Betriebssystem MS-DOS als auch unter MS-OS/2 ablauffähig sind.

Der Microsoft Basic Compiler 6.0 für MS-DOS und MS-OS/2

Der Microsoft Basic Compiler 6.0 unterstützt die Entwicklung von Programmen sowohl für das Betriebssystem MS-DOS und den Real Mode des Prozessors als auch für das Betriebssystem MS-OS/2 und den Real Mode beziehungsweise den unter diesem Betriebssystem erreichbaren Protected Mode des Prozessors. Weitreichende System-Unterstützung, vielseitige Möglichkeiten des Quellcode-Debuggers CodeView, bedeutende Verbesserungen der Programmiersprache Basic und eine Reihe neuer Utilities zeichnen den Basic Compiler von Microsoft in der Version 6.0 aus.

Über den Bereich der Standardleistungen hinaus bietet der Microsoft Basic Compiler 6.0 alle notwendigen Werkzeuge zur Entwicklung einer Anwendung unter den herausragenden Fähigkeiten der Prozessoren 80286 und 80386, wie sie unter Verwendung des Betriebssystems MS-OS/2 erstmals zugänglich werden.

Unter den besonderen Leistungen des neuen Basic-Compilers unter MS-OS/2, findet sich eine SHELL-Funktion sowie die Anweisung OPEN PIPE. Die SHELL-Funktion erlaubt einem Stammprozeß, eine weitere Anwendung in Gang zu setzen, die als Hintergrund-Prozeß asynchron zum Stammprozeß simultan abgearbeitet werden kann. Ein einfaches Beispiel einer derartigen Hintergrund-Anwendung ist beispielsweise in einem Druckprogramm zu sehen, das als Spool-Prozeß im Hintergrund Daten an den Drukker sendet, während der Anwender im Vordergrundprozeß in einem Text-Editor die Arbeit fortsetzen kann. Dieses Konzept der Vorder- und Hintergrund-Arbeit des Prozessors, auch als Parent/Child-Prozeß (Eltern/Kind-Prozeß) bekannt, erfordert die Möglichkeit zum Daten- bzw. Kommunikationsaustausch zwischen beiden Anwendungen. Dieser Austausch wird durch die Anweisung OPEN PIPE sichergestellt, die eine Kommunikationsverbindung zwi-

schen beiden Prozessen hergestellt. Zusätzliche Befehle wie ON SIGNAL, ein Befehl, der signalisierende Informationen aus Hintergrundprozessen sowie aus dem Betriebssystem zum Anwender vordringen läßt, und SLEEP sind weitere Neuerungen innerhalb eines Basic-Compilers.

BASIC-Sprachverbesserungen

An Basic läßt sich in musterhafter Weise der Vorgang einer Sprachen-Weiterentwicklung verfolgen. Um den stets steigenden Anforderungen der Anwendungsprogrammierer noch weiter entgegenzukommen, sollen folgende Verbesserungen erreicht werden:

1. Besserer Zugriff zu Hardware und Betriebssystem.
2. Spracheigenschaften, die der Professionalität der Anwendungen entgegenkommen.

Daraus ergibt sich eine Verbesserung bei der Behandlung von Fehler-Zuständen. Ein Hindernis war in der Vergangenheit die Unfähigkeit dieser Sprache, Fehler insbesondere in modularen Programmen richtig zu behandeln. Durch die in Microsoft Basic 6.0 enthaltene modulorientierte Fehlerbehandlung ist dieser Nachteil ausgeräumt. Microsoft Basic 6.0 bietet nun die Fähigkeit, auf anwendungsspezifische Vorfälle zu reagieren. Dadurch wird es auf einfache Weise möglich, Kontrollprogramme zur Gerätesteuerung und -überwachung in industriellen Anwendungen zu schreiben.

Frühere Microsoft Basic-Versionen fügten der Sprache die Fähigkeit zur strukturierten Programmierung hinzu. Microsoft Basic 6.0 kann Datentypen und Dateien in eigener Definition erzeugen. Analog zu Dateien in Pascal und Strukturen in C erlaubt dies dem Programmierer die Behandlung komplexer Datenstrukturen, wie sie bislang in Basic nicht verfügbar waren. Arrays in der Größenordnung des verfügbaren Speichers werden sowohl unter MS-DOS als auch unter MS-OS/2 verfügbar. Dabei ist die Begrenzung auf 128 Mbyte unter MS-OS/2 lediglich abhängig vom verfügbaren Speicher auf Diskette oder Festplatte, die für die Auslagerung in Form von Swap-Dateien genutzt werden. Maßgerechte Runtime-Systeme lassen sich über die Entscheidung für eine Compilierung zu einem Programm ohne oder mit zusätzlichem Runtime-Modul erreichen. Fällt die Entscheidung zugunsten des zusätzlichen Runtime-Moduls aus, so hat der Programmierer die Möglichkeit, dem Standard-Runtime-Modul eigene Erweiterungen hinzuzufügen. Gleichermaßen steht dem Programmierer die Möglichkeit offen, nicht benötigte Teile des Runtime-Moduls auszulassen und somit die Größe des Moduls zu verringern. Diese Fähigkeiten stehen ebenfalls sowohl für Anwendungen unter MS-DOS als auch für Anwendungen unter MS-OS/2 zur Verfügung.

Zur besonders schnellen Entwicklung von Anwendungen ist im Microsoft Basic 6.0 noch Microsoft QuickBasic 4.0 enthalten. Mit den Vorteilen des Compilers wird unter Microsoft QuickBasic 4.0 eine Interpreter-ähnliche Oberfläche geboten, die Quellcode innerhalb des Editors im Speicher ablegt und durch den integrierten Debugger eine verzögerungsfreie Überprüfung des Programms gestattet. Microsoft QuickBasic 4.0 nimmt dabei die Stellung eines besonders schnellen Prototyp-Entwicklungswerkzeugs ein.

Der optimierende Compiler Fortran 4.1 für MS-DOS und MS-OS/2

Microsoft Fortran 4.1 ist das aktuellste Angebot von Microsoft für die Entwicklung von in der Sprache Fortran geschriebenen Programmen. Der neue Compiler erzeugt sowohl Programmcode für unter dem Betriebssystem MS-DOS ablaufende Programme als auch für Rechner, die mit dem Betriebssystem der Zukunft, MS-OS/2, ausgestattet werden. Programmierer können jetzt sowohl Anwendungen unter dem Speicher-Modell HUGE schaffen als auch Anwendungen von Mini- und Mainframe-Rechnern auf einem Personalcomputer realisieren.

Microsoft Fortran 4.1 bietet volle Unterstützung des Betriebssystems MS-DOS und auch des zukünftigen Betriebssystems MS-OS/2. Während im Real Mode der Prozessoren 80286 und 80386 die bisherige 640-Kbyte-Grenze gilt, wird der Programmierer erstmals in die Lage versetzt, unter Nutzung des Protected Mode der Prozessoren und damit auch unter MS-OS/2, die 640-Kbyte-Grenze zu durchbrechen. Programme dürfen damit auf bis zu 16 Mbyte RAM-Speicher beziehungsweise auf bis zu 1 Gbyte große virtuelle Speicher zugreifen. Das Microsoft Fortran-4.1-Paket enthält einen Incremental Linker, einen im Protected Mode lauffähigen Editor sowie CodeView, den Quellcode-Debugger, in einer ebenfalls im Protected Mode lauffähigen Version.

Microsoft Pascal 4.0 für MS-DOS und MS-OS/2

Microsoft Pascal 4.0 ist ein mächtiges Entwicklungswerkzeug für den professionellen Pascal-Programmierer. Es enthält Entwicklungshilfen für professionelle Anwendungen in lesbarem Code, modularer und übertragbarer Codierung. Die Grundsätze von Programmstrukturierung, System-Erweiterungen, »Flexible Procedure«-Mechanismen und umfangreichen Datenstrukturen befähigen Microsoft Pascal 4.0 insbesondere zur Lösung extrem komplexer Programmierung bei kurzen Entwicklungszeiten und führen zu unter MS-DOS oder MS-OS/2 ablauffähigen Programmen.

In der Kombination von Microsoft Pascal 4.0 und dem Betriebssystem MS-OS/2 wird die im Real Mode der Prozessoren 80286 und 80386 unter dem Betriebssystem MS-DOS vorhandene Begrenzung auf einen verwendbaren Speicher von 640 Kbyte aufgehoben. Programme dürfen nun eine Größe von bis zu 1 Gbyte erreichen, wobei Begrenzungen nur noch durch die verwendeten Speichermedien Diskette oder Festplatte vorgegeben sind, die als Swap-Bereich zum Auslagern beziehungsweise Nachladen von Programmteilen verwendet werden.

Das Programmiersprachen-Paket Microsoft Pascal 4.0 enthält ein BIND-Utility, das die Erzeugung einer einzigen ausführbaren Programm-Datei des Typs .EXE erlaubt, das sowohl unter MS-DOS als auch unter MS-OS/2 gestartet werden darf und in beiden Betriebssystemen gleichermaßen ablauffähig ist.

Microsofts MS-OS/2-Programmierer-Toolkit

Dieses neue Toolkit enthält drei Referenz-Handbücher über MS-OS/2 mit mehr als 900 Seiten Umfang sowie eine Reihe von Software-Hilfswerkzeugen. In Kombination mit jeder der fünf neuen Sprachen-Compiler, die ebenfalls von Microsoft angekündigt wurden, erhalten die Software-Entwickler alle Werkzeuge und Unterlagen zur Erstellung von MS-OS/2-Anwendungs-Software. Folgende Dokumentationen und Hilfswerkzeuge sind Bestandteil des Toolkits:

1. MS-OS/2-»Programmers Reference«
   Dieses Handbuch enthält eine vollständige Beschreibung der MS-OS/2-Systemfunktionen, -Strukturen und -File-Formate.
2. MS-OS/2-Programmier-Werkzeuge
   Dieses Handbuch erklärt den Gebrauch der Software-Werkzeuge, die das Programmierer-Toolkit enthält, sowie Möglichkeiten, MS-OS/2-Anwendungen mit Microsoft-C und MASM zu erstellen.
3. MS-OS/2-»Programmers Learning Guide«
   Dieses Handbuch beschreibt die Basis der MS-OS/2-Programmierung und zeigt Möglichkeiten, Programme zu schreiben, die alle Vorteile des MS-OS/2 nutzen. Neben einer Reihe von anderen Punkten werden in dem Handbuch folgende Themen beschrieben: die Erstellung von C-Programmen, die MS-OS/2-Systemfunktionen zum Öffnen und zum Einsatz von Disk-Files, die Erzeugung von Ablaufpfaden innerhalb eines Programms sowie die Erstellung von dynamisch verbundenen Bibliotheken. Zusätzlich zu der Dokumentation gibt es ein Online-Hilfesystem. Dieses System läßt sich sowohl direkt als auch aus dem Microsoft-Editor heraus aufrufen.

pi/ni

Effektiv programmieren in Microsoft C:

Die Nutzung von Far- und Huge-Datenzeigern

Zeiger sind eine der wichtigsten Neuerungen in der Programmiersprache C, da sie oft der einzigen Weg sind, Ideen klar und präzise darzustellen. Adressen sind in allen Programmiersprachen kritisch, aber die meisten Programmiersprachen verbergen die Details der Adressierung vor Ihnen und somit auch die Information darüber was sich ereignet, wenn Ihr Programm läuft. Nur die Programmiersprache C bringt die Details ans Tageslicht, indem sie Ihnen Zeigeroperationen erlaubt.

In den ersten Tagen von C waren Zeiger 16 Bit lang. Neulinge in der Programmiersprache C mußten sowohl Zeigerarithmetik, als auch die etwas schwierige Notation beim Umgang mit Zeigern lernen. Dafür mußten Sie sich aber nicht allzu viele Gedanken über den verwendeten Computer machen: die PDP 11. Zeigeroperationen werden von der einfachen und konsistenten Architektur der PDP wunderschön unterstützt, aber nur im Datenbereich von 64 Kbyte.

PCs beruhen auf den Mikroprozessoren von Intel, die eine segmentierte Architektur verwenden. Passen Ihre Daten in einen einzigen 64 Kbyte großen Bereich (Microsoft-C-Programme, Small- oder Medium-Speichermodell), dann arbeitet die Zeigerarithmetik effizient und konsistent, wie auf der PDP 11. (Diese Diskussion bezieht sich auf die Segmentierung der Prozessoren 8086, 8088, 186, 188 und den Real-Modus von 286 und 386. Die Segmentierung im geschützten Modus ist unterschiedlich und wird hier nicht erläutert.) Wenn Sie den zusätzlichen Speicherbereich, den Ihnen der PC bietet, nutzen wollen, so müssen Sie den Gebrauch der Far- und Huge-Zeiger lernen.

Microsoft C gestattet Ihnen das Mischen von Datentypen innerhalb eines Programms. Sie können einen Far-Zeiger in einem Programm im Small-Speichermodell ebenso verwenden, wie einen Near-Zeiger in einem Programm im Huge-Speichermodell. Meine Erfahrung beruht auf der Verwendung von Far- und Huge-Datenzeigern in Programmen im Small-Speichermodell. Die Programme, die ich schreibe, sind klein, beinhalten aber einige wenige große Datenstrukturen. Viele meiner Anregungen treffen auf alle Zeiger in Programmen im Compact-, Large- und Huge-Speichermodell zu, da in diesen Speichermodellen Zeiger voreingestellt `far` oder `huge` sind. Alle Programme in diesem Artikel wurden mit dem Microsoft C-Compiler der Version 4.0 im Small-Speichermodell übersetzt.

Neben Zeigern auf Far- und Huge-Daten bietet Microsoft C auch Far- und Huge-Datentypen. Bei der Verwendung von Microsoft C gibt es zwei Möglichkeiten zum Erzeugen eines sehr großen Arrays. Sie können es als Far- oder Huge-Array von Integern deklarieren, oder Sie können einen Zeiger auf eine Far- oder Huge-Integerzahl deklarieren und dann `halloc`, die Allokierfunktion für Huge-Datenbereiche, verwenden.

```
main()
{
        int     *ip, i;


        ip = &i;
        ip += 100;
        printf("ip - &i: %d\n", ip - &i);
        printf("(unsigned)ip - (unsigned)&i: %u\n",
                (unsigned)ip - (unsigned)&i);
}
```

Listing 1: *Zeigerarithmetik.*

Die zweite Methode ist wegen der größeren Flexibilität vorzuziehen. Das Deklarieren eines Huge-Arrays zwingt Sie von vorneherein, die Größe festzulegen. Die Verwendung von `halloc` gestattet Ihnen die Bestimmung der wirklichen Größe zur Laufzeit des Programms. Dieser Artikel beleuchtet hauptsächlich die Verwendung von Zeigern auf Far- und Huge-Daten, obwohl die meisten Aspekte und ein großer Teil der Diskussion über die Effizienz auch Far- und Huge-Arrays betreffen.

Als Vorbedingung für das Verständnis der folgenden Ausführungen müssen Sie zuerst die Verwendung von Zeigern und Casts in C verstanden haben. Sind Ihnen die Grundlagen der beiden Themengebiete bekannt, so ist das Programm in *Listing 1* bestimmt eine Kleinigkeit. Vor dem Weiterlesen sollten Sie einen Blick auf das Programm werfen und die Ausgabe vorhersagen.

100 und 200 sind richtig. Die erste `printf`-Anweisung gibt die Anzahl von Integerzahlen im Intervall zwischen den Stellen an, auf die `ip` zeigt und wo `i` ist, an. Da `ip` die Stelle 100 Integer hinter `i` zugewiesen wurde, ist die Antwort selbstverständlich 100.

Im zweiten printf-Statement werden `ip` und die Stelle von `i` zu `unsigned` umgewandelt und dann abgezogen. Der Cast-Operator ändert nicht das Bitmuster, aber er ändert die Zeiger-Subtraktion in eine normale `unsigned`-Subtraktion. Die zweite Antwort ist 200, da das Bitmuster sich um diesen Betrag unterscheidet.

Wenn Sie fragen, warum 200: Wegen der Größe der Integerzahl von je zwei Byte mal der Anzahl 100. Das Statement `ip += 100` addiert 200 zum Wert von `ip`. Haben Sie das Rätsel gelöst, dann können Sie weiterlesen. Wenn nicht, dann sollten Sie zuerst mehr über Zeigeroperationen lernen, bevor Sie die außergewöhnlichen Zeigeroperationen der Far- und Huge-Zeiger von Microsoft C verstehen können.

Ein Zeiger auf Far-Daten, der gewöhnlich Far-Zeiger genannt wird, ist ein 32 Bit langer Zeiger, dessen arithmetische Operationen aber nur mit 16 Bit durchgeführt werden. (Sehen Sie die Zusammenfassung in *Bild 1.*) Ein Far-Zeiger kann auf jede Speicherstelle im Adressbereich des PCs zeigen. Die 16-Bit-Arithmetikgrenze bedeutet aber, daß der Bereich, auf den er zeigt, nur 64 Kbyte groß sein darf.

	Near	Far	Huge
Größe (bit)	16	32	32
Adressierung	Irgendwo im 64 Kbyte großen Datensegment	Irgendwo im 64 Kbyte großen Datensegment	Irgendwo
Zeigerarithmetik	16 Bit	16 Bit	32 Bit
Microsoft C	Small Medium	Compact Large	Huge

Bild 1: *Datenzeiger in Microsoft C.*

Sie können einen Far-Zeiger verwenden, um auf ein dynamisches Array von 25000 Integerzahlen zuzugreifen, aber Sie können denselben Far-Zeiger nicht für den Zugriff auf ein Array von 25000 Float- oder Long-Elementen verwenden. Dies ergibt sich aus der Tatsache, daß das Integerarray weniger als 64 Kbyte, das Float- oder Double-Array aber mehr als 64 Kbyte Speicherbereich benötigt.

Ein Zeiger auf Huge-Daten wird Huge-Zeiger genannt. Er ist ebenso 32 Bit lang, aber Arithmetikoperationen werden mit allen 32 Bit durchgeführt. Deswegen kann ein Huge-Zeiger überallhin und auf alles zeigen. Der Nachteil ist aber, daß Zeigeroperationen mit Huge-Zeigern langsamer sind und mehr Code benötigen. Die Addition einer Integerzahl zu einem Far-Zeiger ist nicht mehr, als die Addition von zwei Integerzahlen, aber die Addition einer Integerzahl zu einem Huge-Zeiger ist die doppelte Arbeit.

Far- und Huge-Zeiger werden manchmal in Small- und Medium-Programmen verwendet, um Daten außerhalb des 64 Kbyte großen Datenbereichs zu erreichen. Sie können zum Beispiel zum Adressieren des Videospeichers oder zum Zugriff auf dynamisch allokierten Speicher außerhalb des 64 Kbyte großen Bereichs verwendet werden. In Compact- oder Large-Modell-Programmen sind alle Datenzeiger Far-Zeiger, außer wenn Sie diese speziell als `near` oder `huge` deklarieren. In Huge-Modell-Programmen sind alle Zeiger `huge`, außer wenn Sie diese speziell `near` oder `far` deklarieren.

Wenn Sie einen gewöhnlichen Zeiger deklarieren, wie `ip` im Listing 1, dann schreiben Sie einen Stern vor den Namen, um festzulegen, daß `*ip` eine Integerzahl ist. `Ip` muß deshalb ein Zeiger auf eine Integerzahl sein. Um einen Far- oder Huge-Zeiger zu deklarieren, müssen Sie die Worte `far` oder `huge` mit dem Stern verwenden. Sie schreiben also:

```
int far *npfi;
static int huge *hp1,
           huge *hp2;
```

In einer Deklaration oder bei einer Cast-Operation ist das Wort `far` oder `huge` unmittelbar mit dem Stern verbunden. Deshalb erscheinen die Wörter `huge` in der zweiten Deklaration zweimal, einmal für jeden Stern. Beachten Sie, daß der Datentyp und die Speicherklasse, `static Integer`, nur einmal in der zweiten Deklaration stehen.

```
/*
 * Untersuchung der Zeiger
 */
#include       <malloc.h>
#include       <stdlib.h>

main(c, v)
int    c;
char   *v[];
{
   unsigned  nelems;
   char  far *f1, far *f2;

   if(c != 2) {
           printf("%s Anzahl_der_Elemente\n", v[0]);
           exit(1);
   }
   nelems = (unsigned)atol(v[1]);

   f1 = (char far *)_fmalloc(nelems * sizeof(char));
   if(f1 == (char far *)0) {
           perror("_falloc Fehler:");
           exit(1);
   }
   f2 = f1 + nelems;

   printf("Anzahl der Elemente: %u\n", nelems);
   printf("1.%6u == (unsigned)(f2-f1)\n", (unsigned)(f2-f1));
   printf("2.%6ld == (long)(f2-f1)\n", (long)(f2-f1));
   printf("3.%6ld == (unsigned long)(f2-f1\n",
                     (unsigned long)(f2-f1));
   printf("4.%6ld == (long)(unsigned)(f2-f1)\n",
                     (long)(unsigned)(f2-f1));
}
```

Listing 2: *Subtraktion von Zeigern.*

Die erzeugten Zeiger in der obigen Deklaration können Daten im gesamten Speicher erreichen, aber der Speicher für die Zeiger selbst (jeweils 4 Byte) wird im aktuellen Datensegment angelegt. Der erste Zeiger heißt `npfi`, da es ein Near-Zeiger (pointer) auf eine Far-Integerzahl ist. Der Zeiger selbst besteht aus Near-Daten, zeigt aber auf Far-Daten. Die allgemein gebräuchliche Bezeichnung ist Far-Zeiger, da es ein Zeiger auf Far-Daten ist. Sie können einen Near-Zeiger (16-Bit-Zeiger) im Far-Datenbereich, aber nicht im aktuellen Datensegment, durch folgende Deklaration erzeugen:

```
int * far fpni;
```

Der Zeiger in dieser Deklaration wird `fpni` genannt, um Sie zu erinnern, daß es ein Far-Zeiger (pointer) auf eine Near-Integerzahl ist. Der Zeiger (2 Byte) wird im Far-Datensegment gespeichert, obwohl er nur auf eine Integerzahl im aktuellen Datensegment zeigen kann. Diese Deklaration kann nicht in einer Prozedur, nicht einmal innerhalb von `main` erscheinen, da die Deklaration den Compiler anweist, den Zeiger im Far-Datensegment abzuspeichern, wogegen automatische Variablen in einer Prozedur auf dem Stack abgelegt werden. Das Hinzufügen der Speicherklasse `static` jedoch würde es gestatten, die Deklaration innerhalb einer Funktion zu verwenden.

```
/*
 * Untersuchung der Huge-Zeiger
 */
#include        <malloc.h>
#include        <stdlib.h>

main(c, v)
int    c;
char   *v[];
{
   long    nelems;
   long    huge *h1, huge *h2;
   long    l1;
   unsigned long   ul1;

   if(c != 2) {
           printf("%s Anzahl_der_Elemente\n", v[0]);
           exit(1);
   }
   nelems = atol(v[1]);

   h1 = (long huge *)halloc(nelems, sizeof(long));
   if(h1 == (long huge *)0) {
           perror("halloc Fehler: ");
   exit(1);
   }
   h2 = h1 + nelems;
   printf("Anzahl der long Elemente: %lu\n", nelems);
   printf("1.%12u == (unsigned)(h2-h1)\n",(unsigned)(h2-h1));
   printf("2.%12u == (unsigned)(long)(h2-h1)\n",
                     (unsigned)(long)(h2-h1));
   printf("3.%12ld == l1 = h2-h1\n", l1=h2-h1);
   printf("4.%12lu == ul1 = h2-h1\n", ul1=h2-h1);
   printf("5.%12ld == (long)(h2-h1)\n", (long)(h2-h1));
   printf("6.%12lu == (unsigned long)(h2-h1)\n",
                      (unsigned long)(h2-h1));
   printf("7.%12ld == (long)(unsigned)(h2-h1)\n",
                      (long)(unsigned)(h2-h1));
}
```

Listing 3: *Zeigerarithmetik mit Huge-Zeigern.*

```
Zeiger                      0x01000000   0x00ff0010   0x00800800
Höherwertige 16 Bits            0x0100       0x00ff         0x80
Niederwertige 16 Bits              0x0         0x10        0x800
Intel-Segmentierung            0x01000      0x00ff0        0x800
Adreßberechnung                   +0x0        +0x10       +0x800
                            ----------   ----------   ----------
Ergebnis                       0x01000      0x01000      0x01000
```

Bild 2: *Drei verschiedene Zeiger auf die Adresse 1000H.*

Die Wörter `near`, `far` und `huge` sind Microsoft-Erweiterungen der Sprache C. Sie gestatten Microsoft C das Beste an Performance aus der schwierigen Intel-Architektur herauszuholen. Eine Deklaration wie

```
typename far * dataname;
```

erzeugt einen Zeiger im Near-Datensegment, um auf Daten im Far-Datensegment zuzugreifen. Eine Deklaration der Form

```
typename far dataname;
```

erzeugt Daten im Far-Datensegment. Wenn auf das Wort `far` (oder `near` oder `huge`) ein Stern folgt, wird der Zeiger lokal angelegt, kann aber auf entfernte Daten weisen. Wenn das Wort `far` (oder `near` oder `huge`) vor einem Variablennamen steht, dann wird die Variable selbst im Far-(oder Near- oder Huge-)Datensegment angelegt. Der User's Guide des Microsoft C-Compiler enthält einige Beispiele mit Near-/Far-/Huge-Deklarationen.

Das erste Problem bei der Verwendung von Far- oder Huge-Zeigern in einem Small- oder Medium-Programm ist, daß die gewöhnliche Größe für Datenzeiger bei der Parameterübergabe an Funktionen 16 Bit ist. Sie können keinen Far- oder Huge-Zeiger an eine Funktion wie `strlen` übergeben, die einen gewöhnlichen Near-Zeiger erwartet. (Wenn notwendig können Sie eine Bibliotheksfunktion für unterschiedliche Modelle aus Ihrer Bibliothek herausziehen und sie zu Ihrem Programm linken.) Schreiben Sie Ihre eigenen Funktionen, dann seien Sie sorgfältig bei der Deklaration der Parameter wie `far` oder `huge`, wenn Sie beabsichtigen Far- oder Huge-Zeiger zu übergeben. Far- (oder Huge-) und Near-Zeiger als Funktionsparameter können nicht gemischt werden.

Das zweite Problem mit Far- und Huge-Zeigern ist, daß der C-Sprachstandard (sowohl der alte von Kernighan und Ritchie, als auch der kommende ANSI) aussagen, daß der Unterschied zwischen zwei Zeigern eine Integerzahl ist, ebenso wie das Ergebnis des Operators `sizeof`. Das kann Probleme geben, wie das Programm in *Listing 2* zeigt.

Wird das Programm in Listing 2 mit dem Wert 10000 in der Kommandozeile aufgerufen ergibt sich folgendes Ergebnis:

```
Anzahl der Elemente: 10000
1.10000 == (unsigned) (f2-f1)
2.10000 == (long)(f2-f1)
3.10000 == (unsigned long)(f2-f1)
4.10000 == (long)(unsigned)(f2-f1)
```

Es scheint alles in Ordnung zu sein. Verändern Sie aber die Größe der Allokierung auf 40000 - eine vernünftige Größe für ein Zeichenarray, auf das mit einem Far-Zeiger zugegriffen wird - erhalten Sie folgendes Ergebnis:

```
Anzahl der  Elemente: 40000
1.40000 == (unsigned) (f2-f1)
2.-25536 == (long)(f2-f1)
3.-25536 == (unsigned long)(f2-f1)
4.40000 == (long)(unsigned)(f2-f1)
```

Das Problem in der zweiten Zeile der Anzeige ist, daß die Typkonvertierung des Ergebnisses der Subtraktion (eine Signed-Integerzahl) in eine Long-Zahl mit Vorzeichenerweiterung durchgeführt wird, und daher das falsche Ergebnis liefert. Dasselbe Problem kann mit einem gewöhnlichen Array mit mehr als 32767 Elementen auftreten, aber ge-

```
/*
 * Gleichheit von Huge-Zeigern
 */

main()
{
        int     huge *h1, huge *h2;

        h1 = (int huge *)0x01000000;
        h2 = (int huge *)0x00800800;

        printf("%p == %p ? %s\n", h1, h2,
               h1 == h2 ? "yes" : "no");
        printf("(long)(%p - %p) == 0L ? %s\n", h1, h2,
               (long)(h1 - h2) == 0L ? "yes" : "no");
}
```

Listing 4: *Vergleich von Far- und Huge-Zeigern.*

wöhnliche Arrays mit einer Größe von 32 Kbyte sind selten, da es für große Arrays ja Far- und Huge-Zeiger gibt.

Das Problem in der dritten Zeile ist etwas feinsinniger, und es erfordert einen oder zwei Ausflüge in das »Language Reference Manual« des Microsoft C-Compilers, um es zu verstehen. Ein Umwandlung von einer vorzeichenbehafteten Integerzahl (das Ergebnis der Subtraktion von `f2` und `f1`) in eine Unsigned-Long-Zahl wird durch eine Vorzeichenerweiterung in eine Long-Zahl und anschließende Umwandlung in die Unsigned-Long-Zahl durchgeführt. Die Lösung ist einfach. Sie brauchen nur die Signed-Integer-Zahl mit einer Cast-Anweisung in eine Unsigned-Integerzahl wandeln, und sie dann in eine Long-Zahl zu wandeln (in der vierten Zeile).

Listing 3 enthält ein Programm, um die Zeigerarithmetik von Huge-Zeigern zu untersuchen. Da ein Huge-Zeiger auf eine große Datenmenge zeigt, gilt es einige Umwandlungen mehr zu erkunden, als mit Far-Zeigern. Lassen Sie das Programm mit dem Wert 10000 laufen, wird folgendes ausgegeben:

```
Anzahl der long Elemente: 10000
1.59152 == (unsigned) (h2-h1)
2.10000 == (unsigned)(long)(h2-h1)
3.-6384 == l1 = h2-h1
4.4294960912 == ul1 = h2-h1
5.10000 == (long)(h2-h1)
6.10000 == (unsigned long)(h2-h1)
7.59152 == (long)(unsigned)(h2-h1)
```

Die einzigen funktionierenden Fälle sind diejenigen, die das Ergebnis `h2-h1` sofort in eine Long- oder Unsigned-Long-Zahl wandeln. Das Microsoft-Handbuch warnt Sie, daß die Subtraktion von zwei Huge-Zeigern ohne den `cast` nach `long` ein `int` erzeugt, mit dem oben gezeigten Ergebnis. In diesem Fall hat die Typumwandlung nach `long` eine wichtige semantische Bedeutung, die über die gewöhnliche Bedeutung der Typumwandlung hinausgeht. Der Operator `cast` kontrolliert die Präzision des Ergebnisses bei der Subtraktion von Huge-Zeigern.

```
/*
 * Byte Offset von Huge-Zeigern
 */

#include        <dos.h>
#include        <malloc.h>

/* Konvertierung eines MSC-Huge-Zeigers in einen Bytesoffset ab
Adresse 0 */
#define hptr2long(p) (((unsigned long)FP_SEG(p))<<4 +
(unsigned long)FP_OFF(p))

#define NELEM   5

main()
{
    int     huge *hptr[NELEM];
    int     i;

    printf("Huge        Offset\n");
    printf("Zeiger      von 0\n");
    for(i = 0; i < NELEM; i++) {
        hptr[i] = (int huge *)halloc(32768L, sizeof(int));
        printf("%p %71d\n", hptr[i], hptr2long(hptr[i]));
    }
}
```

Listing 5: *Umsetzung von Zeigern auf einen Byteoffset von 0.*

Die andere Tatsache, die Sie vielleicht schon bemerkt haben ist, daß der `cast`, der Ihnen die Subtraktion von zwei Huge-Zeigern erlaubt, ein anderer als derjenige ist, der Ihnen die Subtraktion von zwei Far-Zeigern erlaubt. Seien Sie also doppelt vorsichtig bei der Verwendung dieser gleich aussehenden Zeigertypen, speziell bei Subtraktionen.

Eine andere Schwierigkeit mit Far- und Huge-Zeigern besteht in der Annahme, daß sich Zeiger ordentlich verhalten. Bei der segmentierten Intel-Architektur zeigen viele verschiedene Bitmuster bei Far- und Huge-Zeigern auf die gleiche Speicherstelle. 32 Bit große Zeiger enthalten zwei Teile, die höherwertigen 16 Bit, die in das Segmentregister geladen werden, und die niederwertigen 16 Bit, die den Offset darstellen. In der Intel-CPU werden die Werte der Segmentregister um 4 Bit nach links geschoben und zu den Registern addiert, um die effektive 20-Bit-Adresse zu erhalten. Der Zeiger 0100:0000H (32-Bit-Intel-Zeiger werden oft mit einem Doppelpunkt in der Mitte geschrieben) greift auf die Speicherstelle 1000H zu. Die höherwertigen 16 Bits (0100H) werden vier Bits nach links geschoben (1000H) und dann zu den niederwertigen 16 Bits (0H) addiert. *Bild 2* zeigt zwei andere Zeigerwerte, die auf die Speicherstelle 1000H zeigen. Die Moral von der Geschichte ist, daß der Vergleich zweier Zeiger funktionieren kann oder auch nicht, abhängig vom Bitmuster.

Das Programm in *Listing 4* zeigt die Gefahr des Vergleichs von Far- und Huge-Zeigern. Das erste Statement `printf` gibt `yes` oder `no` aus, je nachdem, ob die zwei Zeiger gleich sind oder nicht. Das zweite `printf` gibt `yes` aus, wenn das Ergebnis der Differenz 0 ist, ansonsten `no`. Mit gewöhnlichen Zeigern (PDP-11-Zeiger oder 16-Bit-Intel-Zeiger) ist das Ergebnis immer gleich.

```
Huge        Offset
Zeiger      von 0

209A:0000   133536
309B:0000   199088
409C:0000   264640
509D:0000   330192
609E:0000   395744
```

Bild 3: *Die Ausgabe des Programms in Listing 5.*

Mit zwei Huge-Zeigern, die auf dieselbe Speicherzelle zeigen, obwohl sie unterschiedliche Bitmuster haben, sieht das Ergebnis folgendermaßen aus:

```
0100:0000 == 0080:0800 ? no
(long)(0100:0000 - 0080:0800) == 0L yes
```

Dieses sonderbare Ergebnis zweier ungleicher Zeiger, die nicht unterschiedlich sind, macht Sinn. Der C-Compiler vergleicht Zeiger durch Vergleich der Bitmuster, aber er muß sie aufwendiger subtrahieren. Der Compiler erzeugt zuerst zwei lineare 20-Bit-Adressen, wie es die CPU hardwaremäßig erledigt, zieht dann die Zeiger voneinander ab und teilt durch die Größe des Datentyps, auf den der Zeiger zeigt. Wenn Sie übliche Dinge mit Ihren Far- und Huge-Zeiger erledigen, wie das Bewegen innerhalb eines allokierten Speicherbereichs, dann funktioniert der Vergleich wunderbar. Etwas unkonventionelles wie selbstkodierte Zeiger oder von der Hardware eingelesene Zeiger oder von anderen Funktionen übergebene zu vergleichen, ist nicht klug. Sie müssen die Zeiger immer mit dem Ausdruck `((long)(p2-p1) == 0)` vergleichen, was erheblich länger dauert, als die bekannte Version `p2 == p1`.

Sie müssen daran denken, daß der Binärwert in einem Far- oder Huge-Zeiger nicht der Offset von der Speicherstelle 0 ist. Statt dessen ist er im Intel-Format Segment:Offset (Listing 2), das addiert werden muß, um den Offset von Speicherstelle 0 zu erhalten. Die meisten Gründe für das Konvertieren eines Zeigers in einen Byteoffset sind schlecht, aber wenn Sie es tun müssen, sehen Sie in *Listing 5* wie es funktioniert. Die Ausgabe des Programms von Listing 5 ist in *Bild 3* zu sehen.

Gewöhnliche Zeiger werden oft nach `unsigned` oder `unsigned long` umgewandelt, wenn ein Byteoffset von 0 gewünscht wird. Dies funktioniert bei Far- und Huge-Zeigern wegen der Segment:Offset-Eigenschaft der Intel-Architektur nicht. Das Makro `hptr2long` ist notwendig für die korrekte Adresse des Zeigers. Die Makros `FP_SEG` und `FP_OFF` zerlegen den Huge-Zeiger in Segment und Offset. Sie sind in `dos.h` definiert.

Lassen Sie uns die Effizienz der Far- und Huge-Zeiger untersuchen. Bevor wir uns den vom Compiler für Far- und Huge-Zeiger generierten Code ansehen, wollen wir uns verdeutlichen, wie dieser aussehen müßte.

```
#include <malloc.h>

#define NELEM   10000

#if defined (HUGE)
#define PTYPE   huge
#define MALLOC halloc(NELEM, sizeof(unsigned))
#define DTYPE   long
#elif  defined (FAR)
#define PTYPE   far
#define MALLOC  _fmalloc(NELEM * sizeof(unsigned))
#define DTYPE   unsigned
#else
#define PTYPE
#define MALLOC  malloc(NELEM * sizeof(unsigned))
#define DTYPE   unsigned
#endif

main()
{
    unsigned PTYPE *head, PTYPE *aspikes, PTYPE *end;
    unsigned time;
    DTYPE diff;

    head = (unsigned PTYPE *)MALLOC;
    aspikes = head;
    end = head + NELEM;
    while ((time = *aspikes) && aspikes++ < end)
            diff = (DTYPE)(end - aspikes);
}
```

Listing 6: *Aufruf häufiger Zeigeroperationen.*

Mit einem Far-Zeiger kann auf jede Speicherstelle zugegriffen werden, indem die 16 höherwertigen Bits in ein Segmentregister (typischerweise ES) und die niederwertigen 16 Bits in ein gewöhnliches Register geladen werden. Für Huge-Zeiger ist diese Operation gleich.

Die Zeigerarithmetik mit Far-Zeigern ist aber erheblich einfacher, als mit Huge-Zeigern. Bei einem Far-Zeiger werden alle arithmetischen Operationen mit den niederwertigen 16 Bits durchgeführt. Alles was der Compiler durchführen muß, sind Operationen mit den niederwertigen 16 Bits, genauso als wären es Near-Zeiger. Bei einem Huge-Zeiger muß der Compiler die Operation zuerst mit den niederwertigen 16 Bits durchführen, dann muß er das Carry- oder Borrow-Bit 12 Bit nach links schieben und dieselbe Operation nochmals mit den höherwertigen 16 Bits durchführen. Dies ist ein großer Aufwand, der oftmals mit Unterstützung von Funktionsaufrufen erledigt wird.

Das Programm in *Listing 6* führt die gebräuchlichsten Zeigeroperationen aus:

- Addition einer Integerzahl zu einem Zeiger
- Dereferenzieren eines Zeigers
- Vergleich zweier Zeiger
- Subtrahieren zweier Zeiger

Der Code kommt aus einem meiner Programme, das Huge-Arrays mit Near- und Far-Zeigern bearbeitet. Die Anweisung `#define` am Anfang des Programms gestattet das leichte Generieren für Near-, Far- und Huge-Zeiger durch die Angabe der Worte `FAR` oder `HUGE` in der Kommandozeile des C-Compilers. Der erzeugte Code für jeden Zeiger ist in *Tabelle 1* dargestellt.

```
#include <dos.h>

char far *
huge2far(n)
char huge *n;
    {
    register unsigned o;
    register unsigned s;
    unsigned d;
    char far *p;

    o = FP_OFF(n);
    s = FP_SEG(n);
    if(o < 0x7ff0) {
            d = (0x8000 - o) & ~0xf;
            o += d;
            s -=(d >> 4);
    }
    else if ( o > 0x8000) {
            d = (o - 0x7ff0) & ~0xf;
            o -= d;
            s += (d >> 4);
    }
    FP_OFF(p) = o;
    FP_SEG(p) = s;
    return(p);
}
```

Listing 7: *Umstellung von Huge-Zeigern.*

Das überraschendste Ergebnis von Tabelle 1 ist, daß der Overhead von Near- und Far-Zeigern ähnlich ist. Dies ändert sich erst bei der Verwendung von Huge-Zeigern. Die aufwendigste Operation bei Huge-Zeigern ist die Subtraktion. Sie benötigt neun Befehle und einen Unterprogrammaufruf.

In meinem Programm muß ich ein Huge-Array mit 50000 Long-Variablen verwalten. Das Programm lief zu langsam, so daß ich die Sache etwas beschleunigen mußte. Die meisten Berechnungen in meinem Programm greifen nur auf kleine Teile innerhalb des Arrays zu, diese sind klein genug, um mit einem Far-Zeiger erreicht zu werden, obwohl das Array so groß ist, daß es mit einem Huge-Zeiger verwaltet werden müßte.

Die Lösung war eine optimale Routine für die Konvertierung eines Huge- in einen Far-Zeiger. Dies bedeutet für meine Applikation, daß der Far-Zeiger einige tausend Male inkrementiert oder dekrementiert werden kann, ohne daß die unteren 16 Bit einen Über- oder Unterlauf verursachen. Die Erfüllung dieser Bedingung setzt voraus, daß die unteren 16 Bit weder zu klein noch zu groß sein dürfen.

Stellen Sie sich einen Huge-Array-Zeiger 2000:0000H vor. Wenn Sie diesen dekrementieren, erniedrigt der Compiler die unteren 16 Bit und borgt sich 1 Bit von den 16 höheren, wodurch sich die korrekte Antwort 1000:FFFEH ergibt. Ist 2000:0000H jedoch ein Far-Zeiger und Sie dekrementieren ihn, dann erniedrigt der Compiler nur die niederwertigen 16 Bit, was zum falschen Ergebnis 2000:FFFEH führt. Far-Zeiger versagen nur bei einem Über- oder Unterlauf. Far-Zeiger deren niederwertige 16 Bits weit von 0 und FFFFH entfernt sind (etwa bei 8000H), sind sicher.

Ich habe ein Routine `huge2far` (*Listing 7*) geschrieben, die einen Huge-Zeiger so ändert, daß sein Offset nahe 8000H liegt. Dies erzeugt einen Far-Zeiger, der jedes Element innerhalb einiger tausend Elemente erreichen kann. Die optimale Struktur eines Near-Zeigers in einer anderen Applikation kann anders sein. Wird zum Beispiel in einem Programm der Zeiger nur inkrementiert, nie dekrementiert, so müßte die Umsetzung den Zeiger in den unteren 16 Bit möglichst zu 0 machen.

Bedenken Sie, daß die Funktion `huge2far` nicht für alle möglichen Zeiger richtig arbeitet. Es gibt zum Beispiel nur einen Zeiger, der die Stelle 0 adressiert, nämlich 0000:0000H. Normalerweise zeigen aber Huge-Zeiger über das Programm und die Daten hinaus - und für diese Bereiche arbeitet die Funktion richtig.

Ich verwende in meinem Programm Huge-Zeiger für die Verwaltung des Arrays, aber sobald eine intensive lokale Bearbeitung ansteht, wandle ich den Huge-Zeiger in einen Far-Zeiger, führe die Berechnung durch und falls der Ergebniszeiger weiterverwendt werden muß, wandle ich ihn mit einer Cast-Operation in einen Huge-Zeiger zurück. Es lohnt nicht, für einen einzigen Zugriff den Huge- in einen Far-Zeiger zu wandeln, wollen Sie aber die nächsten 1000 Zahlen in dem Array aufsummieren, so können Sie den Huge-Zeiger in einen Far-Zeiger wandeln und diesen dann für den Zugriff auf die 1000 nächsten Arrayelemente verwenden. Die Zeit, die ich hierdurch in meinem Programm einspare, ist bemerkenswert.

Der Code in Listing 7 liefert noch mehr Ideen für das Schreiben effizienterer Programme mit Huge-Zeigern. Bedenken Sie, daß die aufwendigste Operation die Subtraktion oft in Schleifen für das Abbruchskriterium verwendet wird. Programmteile wie der folgende werden oft in C-Programmen verwendet (`p`, `q`, `head` und `tail` sind Zeiger):

```
while(p++ < q)
    .
for(p = head; p < tail; p++)
    .
```

Handelt es sich um Huge-Zeiger, wird bei jedem Durchgang viel Zeit für den Vergleich benötigt. Eine schnellere Lösung vermeidet den Vergleich bei jedem Durchgang, wie im folgenden gezeigt:

```
cnt = p - q;
while(p++, cnt--)
    .
for(cnt = tail - p, p = head; cnt > 0; cnt--, p++)
    .
```

Sie müssen natürlich aufpassen, daß der Wert von `p` in der Schleife nicht geändert wird.

Kaare Christian/ni

C	NEAR	FAR	HUGE
end = head + NELEM;	add ax,20000 mov [bp-6],ax ;end	add ax,20000 mov [bp-8],ax ;end mov [bp-6],dx	mov ax,20000 cwd add ax,[bp-18] ;head adc dx,0 mov cl,OFFSET _AHSHIFT shl dx,cl add dx,[bp-16] mov [bp-10],ax ;end mov [bp-8],dx
time = *aspikes	mov bx,[bp-8] ;aspikes mov ax,[bx] mov [bp-4],ax ;time	les bx,[bp-12] ;aspikes mov ax,es:[bx] mov [bp-4],ax ;time	les bx,[bp-14] ;aspikes mov ax,es:[bx] mov [bp-6],ax ;time
aspikes++ < end	mov ax,[bp-8] ;aspikes add WORD PTR [bp-8],2 cmp ax,[bp-6] ;end	mov ax,[bp-12] ;aspikes mov dx,[bp-10] add WORD PTR [bp-12],2 cmp ax,[bp-8] ;end	mov ax,2 cwd add ax,[bp-14] ;aspikes adc dx,0 mov cl,OFFSET _AHSHIFT shl dx,cl add dx,[bp-12] mov cx,[bp-14] ;aspikes mov bx,[bp-12] mov [bp-14],ax ;aspikes mov [bp-12],dx cmp cx,[bp-10] ;end
diff = (DTYPE) (end - aspikes);	mov ax,[bp-6] ;end sub ax,[bp-8] ;aspikes sar ax,1 mov [bp-2],ax ;diff	mov ax,[bp-8] ;end sub ax,[bp-12] ;aspikes sar ax,1 mov [bp-2],ax ;diff	push dx push ax push WORD PTR [bp-8] push WORD PTR [bp-10];end sar dx,1 rcr ax,1 mov [bp-4],ax ;diff mov [bp-2],dx

Tabelle 1: *C-Codeerzeugung für Zeigeroperationen in den Speichermodellen Near, Far und Huge.*

Lesefutter für Wissensdurstige:

Know-How-Sammlungen

Bereits vor 1½-Jahren gab Microsoft Press ein Werk mit dem Titel »The MS-DOS Encyclopedia« heraus, das jedoch wegen zu vieler Fehler schnell wieder vom Markt genommen wurde. Das Projekt war damit jedoch nicht ganz gestorben; Microsoft Press beauftragte den bekannten Fachautor Ray Duncan mit der Betreuung einer überarbeiteten Version, die nun seit Anfang des Jahres erhältlich ist. Neben Ray Duncan arbeiteten zahlreiche weitere Fachautoren und Microsoft-Mitarbeiter an diesem Werk mit, darunter so bekannte Leute wie Charles Petzold, der den Lesern des *Microsoft System Journals* bereits aus zahlreichen Artikeln über Windows- und OS/2-Programmierung bekannt sein dürfte, Gordon Letwin, der Chefentwickler für Systemsoftware bei Microsoft und zahlreiche auch in Deutschland nicht ganz unbekannte Autoren von Microsoft-Press-Büchern, wie Thom Hogan, Van Wolverton und Jo-Anne Woodcock.

Laut Wörterbuch bedeutet Enzyklopädie »Gesamtheit des Wissens« und genau das war das Ziel dieser MS-DOS-Enzyklopädie: Alles für Programmierer und Anwender Wissenswerte über MS-DOS in einem Buch zu sammeln. Das Projekt ist den Autoren geglückt, es dürfte wohl kaum eine Frage zu MS-DOS geben, die in diesem Buch unbeantwortet bleibt. Sehr gut bekommt dem Werk die Beschränkung auf eine klar umrissene Zielgruppe: Programmierer; während andere Bücher viel Platz auf Grundlagenthemen wie Aufbau von Bits & Bytes verschwenden, kommt die MS-DOS-Enzyklopädie bei allen Themen gleich ohne Umschweife zur Sache.

Die Enzyklopädie hat einen Umfang von 1570 großformatigen Seiten und gliedert sich in fünf Teile und 15 Anhänge. Teil 1 beschreibt die Entwicklung von MS-DOS bis zur Version 3.2. Teil 2 ist in fünf Abschnitte aufgeteilt, die sich mit der Struktur, der Programmierung, der Anpassung, der zukünftigen Entwicklung und Programmiertools unter MS-DOS beschäftigen. In Teil 3 werden alle Anwenderbefehle, Konfigurationsbefehle und Gerätetreiber von MS-DOS detailliert beschrieben. Im vierten Teil folgen dann die Programmierutilities von MS-DOS wie CREF, EXE2BIN, LINK und die Debugger (auch Symdeb und CodeView). Teil 5 schließlich enthält eine Beschreibung aller Systemaufrufe von MS-DOS. In den 15 Anhängen geht es um Fehlermeldungen, Fehlercodes, Zeichensätze, Tastencodes, Datenstrukturen (FCB, PSP), Dateiformate (EXE/OBJ) und BIOS-Aufrufe. Behandelt werden die MS-DOS Versionen bis 3.2, die Unterschiede und Erweiterungen der MS-DOS-Version 3.3 werden in einem Anhang beschrieben.

Die Enzyklopädie ist so umfangreich, daß sich die wichtigsten behandelten Themen nur kurz in Stichworten umreißen lassen: Speicheraufteilung, Dateisystem, Aufbau von Programmen (EXE und COM), Ein-/Ausgabe, interruptgesteuerte Kommunikation, Dateiverwaltung, Zugriff auf Verzeichnisse und Label, Speicherverwaltung, die EXEC-Funktion, die Entwicklung von TSR-Utilities (Terminate and Stay Resident - speicherresidente Programme), Fehlerbehandlung, Interruptbehandlung, Filter, Programmierung von Gerätetreibern, Kompatibilität zu OS/2, ein Überblick über Microsoft Windows, Debuggen unter MS-DOS, Objekt-Module und der Linker u.v.m.

Die Themen sind immer umfassend und übersichtlich behandelt. Der einzige Nachteil: Trotz des großen Umfangs wird wirklich nur MS-DOS beschrieben; für ausführliche Informationen über die Programmierung via BIOS oder direkten Hardwarezugriff (besonders wichtig für die Bildschirmsteuerung) muß man weiterhin auf andere Werke zurückgreifen. Doch trotzdem ist das Buch für professionelle Programmierer sehr zu empfehlen. Wer es sich bei dem deutschen Preis von ca. DM 300,- (bis 420,-) leisten kann, sollte unbedingt zugreifen.

Eine deutsche Überstzung der MS-DOS-Enzyklopädie ist im Vieweg-Verlag in Vorbereitung. Sie wird jedoch voraussichtlich nicht in einem Band, sondern in ausgewählten Themenbänden auf den Markt kommen, so daß man hierbei wahrscheinlich billiger wegkommen kann.

Ray Duncan (General Editor): »The MS-DOS Encyclopedia«, Redmond: Microsoft Press, 1988; 1570 Seiten; ISBN 1-55615-049-0; $134.95 (ca. DM 300,-). 5¼- oder 3½-Zoll-Diskette zum Buch $54.95 (inkl. Versandkosten).

Ein deutsches Werk, daß den Intentionen der MS-DOS-Enzyklopädie in Umfang und Ausführlichkeit gleichkäme, gibt es in noch nicht, doch es gibt gute Ansätze dazu. Ein Buch, daß mir positiv aufgefallen ist, ist »PC Intern« von Michael Tischer. Es geht darin hauptsächlich um die systemnahe Programmierung auf PCs, wobei im Gegensatz zur Enzyklopädie jedoch auch besonders auf das BIOS und direkte Hardwarezugriffe eingegangen wird. Das Buch ist mit 767 Seiten sehr umfangreich und enthält neben den Beschreibungen der Programmierthemen auch ausführliche Programmbeispiele in Assembler, BASIC, C und Pascal. Erfahrungen mit diesen Programmiersprachen werden vorausgesetzt. Alle Beispielprogramme sind ausführlich kommentiert. Die Beispiele sind auf einer separaten Diskette für DM 39,- erhältlich.

Der Autor beschreibt im Vorwort, wie es zu diesem umfangreichen Werk kam: Da es auf dem Markt einfach keine umfassende Beschreibung der Programmierung auf PCs (mit den Bereichen Hardware, BIOS und DOS) gab, wie er sie aus dem Heimcomputerbereich kannte, beschloß er, so etwas selbst zu schreiben. Und der Erfolg gibt ihm recht: Michael Tischer beschreibt in diesem Buch in einfachem und gut verständlichem Deutsch alle wichtigen Aspekte der Systemprogrammierung auf PCs und es ist überall zu sehen, daß der Autor etwas von der Sache versteht und auch schon selbst umfangreiche Programmierprojekte durchgezogen hat. Die Beispielprogramme sind in der Regel sehr realitätsnah und von hohem Wiederverwendungswert. Mit ausführlichen Listings beschrieben werden unter anderm fol-

gende Themen: BASIC-Assembler-Kopplung; Verzeichnisanzeige; Dump; Gerüst eines Zeichen-Gerätetreibers; Gerätetreiber für RAM-Disk (Blockgerät); Bildschirmzugriff über BIOS; PC-Konfiguration feststellen; Diskettenmonitor; Tastaturabfrage; sichere Druckausgabe; Codewandlung für Druckausgabe; Bildschirmzugriffe (MDA, CGA und Hercules); Bildschirm-Hardcopy.

In drei zusammen 170 Seiten starken Anhängen werden alle Hardware-, DOS- und BIOS-Interrupts ausführlich beschrieben, so daß sich das Buch auch als Nachschlagewerk sehr gut eignet. Es gibt wenig auszusetzen an diesem Buch. Vermißt habe ich lediglich Beschreibungen der Programmierung der Bildschirmadapter EGA und VGA, die (noch?) nicht enthalten sind.

Bei seinem großen Umfang und Informationsreichtum ist »PC Intern« für jeden Programmierer eine sinnvolle Bereicherung seiner Bibliothek, besonders bei dem durchaus akzeptablen Preis von DM 69,-. Aber besonders Privat- und Hobby-Programmieren möchte ich das Buch sehr ans Herz legen, denn sie werden für die systemnahe Programmierung wahrscheinlich lange Zeit nicht mehr brauchen, als dieses eine Werk.

Michael Tischer: »PC Intern - Systemprogrammierung«, Düsseldorf: Data Becker, 1987; 767 Seiten; ISBN 3-89011-235-8; DM 69,-. 5¼- oder 3½-Zoll-Diskette zum Buch DM 39,-.

Neben *Byte*, *PC Tech Journal* und *Computer Language* ist das *Dr. Dobb's Journal of Software Tools* die wichtigste US-Zeitschrift für PC-Programmierer. Diese Zeitschrift ist die älteste spezielle Programmiererzeitschrift und immer noch eine der interessantesten. Beliebt wurde und ist diese Zeitschrift vor allem wegen seiner umfangreichen Listings. Andere US-Zeitschriften wie *Byte* und *Computer Language* haben schon vor einiger Zeit aufgehört, lange Listings abzudrucken, sie sind meist nur noch über Modem oder Bestellung auf Diskette erhältlich, was beides aus Deutschland nicht immer ohne Probleme funktioniert, oft lange dauert und meist mit hohen Kosten verbunden ist. *Dr. Dobb's* druckt seine Listings weiterhin ab, so daß die Zeitung für Programmierer auch allein, ohne Mailbox oder Diskettenbestellung, nützlich ist.

Es ist nur wenig bekannt, daß das *Dr. Dobb's Journal of Software Tools* im Verlag M&T-Publishing herausgegeben wird, der eine Tochtergesellschaft des deutschen Markt & Technik Verlags ist. Deshalb ist das *Dr. Dobb's Journal* auch in Deutschland relativ einfach und auch im Abonnement zu bekommen.

Das *Dr. Dobb's Journal* existiert bereits seit 1976 und viele der Programme, die darin früher einmal beschrieben und abgedruckt wurden, sind auch heute noch aktuell und enthalten nützliche Routinen. Deshalb sind bei Markt & Technik auch alle bisher veröffentlichten Ausgaben in Jahresbänden zum Preis von DM 95,- erhältlich. Alle 11 Bände zusammen (1976-1986) kosten DM 830,-.

Ich kann diese Zeitschrift, und zwar sowohl die älteren als auch die aktuellen Ausgaben nur empfehlen. Ich selbst habe aus dieser Zeitschrift sehr viel gelernt und glaube, daß das auch anderen so gehen wird.

Dr. Dobb's Journal of Software Tools for Advanced Programmers: Jahresbände (bisher 11 Bände erschienen (1976-1986), Haar: Markt & Technik; Best.Nr.: 90391-90400 und 90812; je Band DM 95,-; alle 11 Bände zusammen (Best.Nr. 90813) DM 830,-.

Zahlreiche Artikel aus *Dr. Dobb's Journal* wurden von den Autoren als Themenbände herausgegeben. Die interessantesten stammen von J.E. Hendrix und A. Holub. J.E. Hendrix wurde vor allem durch seinen Small-C-Compiler bekannt, den er - wie auch alle seine anderen Projekte - mit komplettem Quellcode veröffentlichte. Small-C ist ein Subset von C; Programme in Small-C lassen sich relativ einfach in Microsoft C umsetzen. Diese Software wurde bereits im ersten *Microsoft System Journal* vom November 1987 vorgestellt (S. 35). Darüber hinaus hat J.E. Hendrix noch eine Sammlung von Tools in Small-C und einen 8080- und Z80-Makro-Assembler veröffentlicht, die beide sehr zu empfehlen sind. Seine neueste Entwicklung ist »Small-Windows«, ein Paket von C-Routinen für die Bildschirm- und Fensterverwaltung auf PCs. Diese Routinen werden in Small-C und in Microsoft-C geliefert und sind eine solide Basis für eine komfortable Benutzerschnittstelle in C.

Allen Holub ist der Autor der Kolumne »C-Chest« in Dr. Dobb's Journal. In dieser Kolumne hat er in zahlreichen Fortsetzungen mehrere große Projekte veröffentlich, die auch gesammelt erhältlich sind. Am interessantesten für die Leser des *Microsoft System Journals* sind sein Textformatierer, der in der Leistung in etwa dem UNIX-Textformatierer NROFF entspricht, seine UNIX-artige Benutzerschnittstelle für MS-DOS und eine umfangreiche Utility-Sammlung. Auch wenn man diese Programme nicht als solche benötigt, sind sie sehr zu empfehlen, da sie sehr viele allgemein nützliche Unterroutinen enthalten, die man häufig sehr gut in eigenen Programmen verwenden kann.

J.E.Hendrix: »Small Windows«, Haar: Markt & Technik, 1987; 95 Seiten, inkl. Diskette; ISBN 3-89090-831-4; DM 79,-.
J.E.Hendrix: »Small Tools 1.2«, Haar: Markt & Technik, 1987; 81 Seiten, inkl. Diskette; ISBN 3-89090-810-1; DM 79,-.
J.E.Hendrix: »Small Mac«, Haar: Markt & Technik, 1987; 71 Seiten, inkl. 2 Disketten; ISBN 3-89090-811-X; DM 79,-.
A. Holub: »Nr: an Nroff-like Text Formatter for MS-DOS«, Haar: Markt & Technik, 1987; ca 100 Seiten, inkl. Diskette; ISBN 3-89090-832-2; DM 79,-.
A. Holub: »On command: Writing a Unix-like Shell for MS-DOS«, Haar: Markt & Technik, 1987; 319 Seiten, inkl. Diskette; ISBN 3-89090-830-6; DM 99,-.
A. Holub: »Dr. Dobb's Journal/UTIL«, Haar: Markt & Technik, 1987; 76 Seiten, inkl.Disk; ISBN 3-89090-829-2; DM 79,-.

Nützliche Unterroutinen für jedes C-Programm:

Bildschirmsteuerung und Zeileneditierung

Wenn ein C-Programm komfortabler zu bedienen sein soll, als es die Standard-Bibliotheksfunktionen zulassen, muß man sich eigene Bibliotheken aufbauen. Zu den fast immer benötigten Routinen gehören solche für die Bildschirmsteuerung und die Zeileneditierung. Allen Holub stellt zwei elegante Lösungen dafür vor.

Da die beiden hier vorgestellten C-Programmdateien recht lang sind, mußte der Text kurz gehalten werden. Die einzelnen Routinen, besonders von `VBIOS.C`, sind jedoch so ausführlich kommentiert, daß weitere Einzelbeschreibungen auch nicht notwendig sind. Statt dessen hier ein kurzer Überblick über die Möglichkeiten von `efgets()`, der Routine für die komfortable Editierung von Eingaben.

Die Routine `efgets()` gleicht in vielen Punkten der Bibliotheksfunktion `fgets()`. Es gibt jedoch mehrere wichtige Unterschiede: Nach erfolgreicher Eingabe wird ein Zeiger auf das Ende des Eingabepuffers zurückgegeben; die Eingabe von mehrzeiligen Texten wird unterstützt; der wichtigste Unterschied macht sich bemerkbar, wenn die Eingabe von `Stdin` erfolgt, dann können nämlich folgende Tasten zur Editierung verwendet werden:

[←] bewegt den Cursor nach links, ohne etwas zu löschen.
[→] bewegt den Cursor nach rechts.
[Ctrl][←] bewegt den Cursor zum vorherigen Wort.
[Ctrl][→] bewegt den Cursor zum nächsten Wort.
[Home] bewegt den Cursor an die Position, wo sich der Cursor beim Aufruf von `efgets()` befand.
[End] positioniert den Cursor hinter das am weitesten rechts stehende Zeichen im Eingabefeld.
[Esc] löscht den Eingabepuffer ohne die Zeichen auf dem Bildschirm zu löschen und kehrt mit -1 zum aufrufenden Programm zurück.
[Del] löscht das Zeichen unter dem Cursor und verschiebt den Rest der Zeile um eine Position nach links.
[Ins] schaltet zwischen Überschreiben und Einfügen um.
[Ctrl][H] bewegt den Cursor um ein Zeichen nach links und löscht das Zeichen unter dem Cursor.
[Ctrl][X] löscht die ganze Zeile und den Puffer, es kann jedoch weiter editiert werden.
[Return] beendet die Eingabe der Zeile. `Efgets()` übergibt einen Zeiger auf das Ende des Puffers (\0) an die aufrufende Routine oder NULL bei EOF.

Jedes andere Zeichen größer 32 wird als gültiges Zeichen akzeptiert und in den Eingabepuffer geschrieben.

Allen Holub

Die beiden Listings sind ein Auszug aus dem Buch »C-Chest and Other C Treasures from Dr. Dobb's Journal« von Allen Holub, erschienen bei M&T Books, Redwood City, Kalifornien, 1987. Eine deutsche Übersetzung ist beim Verlag Markt & Technik in Vorbereitung, mit dessen freundlicher Genehmigung der Abdruck erfolgt. jü

```
#include <stdio.h>
#include <dos.h>    /* (Microsoft file) includes for int86() */

/*  VBIOS.C: Various cursor and i/o routine using
 *  the bios interrupts (see below for greater detail):
 *------------------------------------------------------------
 *  Copyright (C) 1987 Allen I. Holub. All rights reserved.
 *------------------------------------------------------------
 * Externally accessable routines:
 *
 * int   vb_getpage ()          Get active video page #
 * void  vb_putchar (c)         write a single character
 * void  vb_getchar (c)         get a key from the bios.
 * void  vb_puts    (s, move)   write a string
 * void  vb_replace (c)         write char w/o moving cursor
 * int   vb_inchar  (attrib)    Get character & attribute
 *
 * void  vb_setcur  (posn)      Set curpos as int on cur page
 * int   vb_getcur  ()          Get curpos as int from cur page
 * void  vb_ctoyx (y,x)         Set cursor position to (y,x)
 * void  vb_getyx (&y, &x)      Get cursor position
 *
 * int   vb_iscolor()           color monitor installed
 * void  vb_cursize (top,bot)   Set cursor size
 * void  vb_blockcur()          make a block cursor
 * void  vb_normalcur()         revert to a normal cursor
 *
 * void  vb_scroll(l,r,t,b,a)   Scroll region
 */


/*----------------------------------------------------------*/
extern int      int86( int, union REGS *, union REGS *);


/*----------------------------------------------------------*/
#define VIDEO_INT     0x10     /* Video interrupt          */
#define KB_INT        0x16     /* Keyboard interrupt       */

#define CUR_SIZE      0x1      /* Set cursor size          */
#define SET_POSN      0x2      /* Modify cursor posn       */
#define READ_POSN     0x3      /* Read current cursor posn */
#define WRITE         0x9      /* Write character          */
#define WRITE_TTY     0xe      /* Write char & move cursor */
#define GET_VMODE     0xf      /* Get video mode & disp pg */


/*----------------------------------------------------------*/
static union REGS        Regs;      /* Used to talk to DOS */
static int               Attribute; /* Current attribute   */


/*----------------------------------------------------------*/
void    vb_scroll( x_left, x_right, y_top, y_bottom, amt )
{
        /* Scroll the indicated region on the screen.
         * If amt is negative, scroll down.
         */

        if( amt < 0 )
        {
            Regs.h.ah = 7    ;
            Regs.h.al = -amt ;
        }
        else
        {
            Regs.h.ah = 6    ;
            Regs.h.al = amt  ;
        }

        Regs.h.bh = 0x07     ;
        Regs.h.cl = x_left   ;
        Regs.h.ch = y_top    ;
        Regs.h.dl = x_right  ;
        Regs.h.dh = y_bottom ;
        int86(0x10, &Regs, &Regs);
}
```

```
/*-------------------------------------------------------*/
int     vb_inchar( attrib )
int     *attrib;
{
        /* Return the character at the current cursor
         * position and, if attrib is non-NULL, put the
         * attribute there. Note that vb_getpage() will mess
         * up the fields in the Regs structure so it must
         * be called first.
         */

        Regs.h.bh = vb_getpage() ;
        Regs.h.ah = 8 ;

        int86( VIDEO_INT, &Regs, &Regs );

        if( attrib )
                *attrib = Regs.h.ah & Øxff ;

        return( Regs.h.al & Øxff );
}


/*-------------------------------------------------------*/
int     vb_getpage()
{
        /* Returns the currently active display page number
         */

        Regs.h.ah = GET_VMODE;
        int86( VIDEO_INT, &Regs, &Regs );

        return (int) Regs.h.bh ;
}


/*-------------------------------------------------------*/
void    vb_cursize( top_line, bot_line )
{
        /* Scan lines are numberd Ø at the top and 7 at the
         * bottom on the color card. On the monochrome card
         * they're Ø-12. If top & bot are reversed you'll
         * get a 2 part cursor. Top_line determines the
         * position of the top scan line of the cursor,
         * bot_line is the bottom. A normal cursor can be
         * created with vb_cursize(6,7). Cursize(Ø,7) will
         * fill the entire area occupied by a character.
         * Cursize(Ø,1) will put a line over the character
         * rather than under it.
         */

        Regs.h.ch = top_line ;
        Regs.h.cl = bot_line ;
        Regs.h.ah = CUR_SIZE ;
        int86( VIDEO_INT, &Regs, &Regs );
}


/*-------------------------------------------------------*/
int vb_iscolor()   /* Returns true if a color card is active */
{
        Regs.h.ah = GET_VMODE ;
        int86( VIDEO_INT, &Regs, &Regs );
        return( Regs.h.al != 7 );
}

void vb_blockcur()      /* Make the cursor a block curser */
{
        vb_cursize( Ø, vb_iscolor() ? 7 : 12 );
}

void vb_normalcur()       /* Make it an underline cursor */
{
        if( vb_iscolor() )
                vb_cursize(  6,  7 );
        else
                vb_cursize( 11, 12 );
}
```

```
/*-------------------------------------------------------*/
void    vb_setcur( posn )
int     posn;
{
        /* Modify current cursor position. The top byte of
         * "posn" value holds the row (y), the bottom byte,
         * the column (x). The top-left corner of the screen
         * is (Ø,Ø). Pagenum is the video page number. Note
         * that vb_getpage() will mess up the fields in the
         * Regs structure so it must be called first.
         */
        Regs.h.bh = vb_getpage() ;
        Regs.x.dx = posn         ;
        Regs.h.ah = SET_POSN     ;
        int86( VIDEO_INT, &Regs, &Regs );
}

int     vb_getcur()
{
        /* Get current cursor position. The top byte of the
         * return value holds the row, the bottom by the
         * column. Pagenum is the video page number. Note
         * that vb_getpage() will mess up the fields in the
         * Regs structure so it must be called first.
         */
        Regs.h.bh = vb_getpage() ;
        Regs.h.ah = READ_POSN    ;
        int86( VIDEO_INT, &Regs, &Regs );
        return( Regs.x.dx );
}

/*--------------------------------------------------------
 * vb_cotyx() and vb_getyx also get the cursor position.
 * They use x and y values, however.
 */
void    vb_ctoyx ( y, x )
{
        vb_setcur( (y << 8) | (x & Øxff) );
}

void    vb_getyx( yp, xp )
int     *yp, *xp;
{
        register int    posn;

        posn = vb_getcur();
        *xp  = posn          & Øxff ;
        *yp  = (posn >> 8) & Øxff ;
}

/*-------------------------------------------------------*/
vb_replace(c)
{
        /* Overwrite the character at the current cursor
         * position without moving the cursor.
         */

        Regs.h.ah = 1Ø ;
        Regs.h.al = c;          /* write c              */
        Regs.h.bl = ØxØ7;       /* Normal characters    */
        Regs.h.bh = Ø;          /* Display page Ø       */
        Regs.x.cx = 1;          /* # of times to write  */

        int86( VIDEO_INT, &Regs, &Regs );
}

/*-------------------------------------------------------*/
vb_putchar( c )
{
        /* Write a character to the screen in TTY mode.
         * Only normal printing characters, BS, BEL, CR and
         * LF are recognized. The cursor is automatically
         * advanced and lines will wrap.
         */

        Regs.h.bl = ØxØ7;
        Regs.h.al = c;
        Regs.h.ah = WRITE_TTY ;
        int86( VIDEO_INT, &Regs, &Regs );
}
```

```
/*--------------------------------------------------------*/
vb_puts( str, move_cur )
register char   *str;
{
        /* Write a string to the screen in TTY mode. If
         * move_cur is true the cursor is left at the end
         * of string. If not the cursor will be restored to
         * its original position (before the write).
         */

        register int    posn;

        if( !move_cur )
                posn = vb_getcur();

        while( *str )
                vb_putchar( *str++ );

        if( !move_cur )
                vb_setcur( posn );
}

/*--------------------------------------------------------*/
int     vb_getchar()
{
        /* Get a character with a direct video bios call.
         * This routine can be used to complement stderr as
         * it can be used to get characters from the keyboard,
         * even when input redirected. The typed character
         * is returned in the low byte of the returned
         * integer, the high byte holds the auxillary byte
         * used to mark ALT keys and such. See the Technical
         * Reference for more info.
         */

        Regs.h.ah = 0 ;
        int86( KB_INT, &Regs, &Regs );
        return( (int)Regs.x.ax );
}

/*--------------------------------------------------------*/
#ifdef MAIN

main()
{
        vb_replace( 'X' );
        vb_putchar('\n');
        vb_putchar('\r');
}

#endif
```

Listing 1: VBIOS.C, *Routinen zur Bildschirmsteuerung.*

```
#include <stdio.h>
#include <signal.h>


/* EFGETS.C        An editing version of efgets.
 *
 * Compile with:  cl -D DEBUG efgets.c vbios.c
 *
 * Copyright (C) 1985,1987 Allen I. Holub. All rights reserved.
 *
 *-----------------------------------------------------------
 * Known bugs:  When there's a character in the rightmost
 *              position in the buffer, and you delete the
 *              character to its left, go_end goes one space
 *              less than it should.
 *-----------------------------------------------------------
 */

                                        /* Public functions */
void    init_egets ( char *buf, int bufsize               );
int     fmhist     ( char *buf, int bufsize, int offset   );
char    *egets     ( char *buf, int bufsize               );
char    *getl      ( char *buf, int maxline, FILE *fp     );
char    *efgets    ( char *buf, int maxline, FILE *fp     );


                                        /* Local static functions */
int     getkey     ( void );
int     pchar      (int c, int scroll_ok);
void    ptail      ( char *bp, char *end, int move );
int     addchar    ( int c );
int     left       ( void );
void    left_word  ( void );
int     right      ( int scroll_ok );
void    right_word( void );
void    tab        ( void );
void    delete     ( void );
void    era_left   ( void );
void    go_home    ( void );
void    era_line   ( void );
void    go_end     ( void );
void    change_insertmode( void );


/*-----------------------------------------------------------*/

typedef int     (*PFI)(); /* Pointer to subr. returning int */

extern int   bdos    ();
extern void  *memmove ();


/* Some editors (vedit) pad out the last sector with ^Z rather
 * than making the file the correct lenghth. We have to test
 * for this explicitly because fgetc() doesn't recognize
 * ^Z as EOF.
 */

#define CTL_Z           0x1a


/* Values returned from DOS when cursor keys etc. are hit:
 */

#define _LEFT           75
#define _RIGHT          77
#define _CTL_LEFT       115
#define _CTL_RIGHT      116
#define _INS            82
#define _DEL            83
#define _HOME           71
#define _END            79
#define _UP             72
#define _DOWN           80


/* The above are mapped as follows by getkey
 */

#define LEFT            0x8000
#define RIGHT           0x8001
#define CTL_LEFT        0x8002
#define CTL_RIGHT       0x8003
#define INS             0x8004
#define DEL             0x8005
#define HOME            0x8006
#define END             0x8007
#define UPKEY           0x8008
#define DOWNKEY         0x8009

#define BDOS_IN   8     /* raw (non echo) input function */

#define CNTL_C  0x03            /*  ^C  */
#define CNTL_Z  0x1a            /*  ^Z  */
#define ESC     0x1b            /*  ^[  */
#define CAN     0x18            /*  ^X  */

#define min(a,b)        ((a) < (b) ? (a) : (b))
```

```
/*-------------------------------------------------------------
 * Cursor manipulation routines:
 *      UP:     Update the cursor position by decrementing the
 *              column position.
 *      COL     Extract the column from cursor
 *      ROW     Extract the row from cursor
 *      BELL    ring the bell
 *      NEWLINE Put the cursor on the left edge of next line
 *              Note that this macro can cause problems if
 *              followed by a semicoln-else
 */
#define UP(cur)         ((cur) -= Øx1ØØ)
#define ROW(cur)        (((cur)>>8) & Øxff)
#define COL(cur)        ( (cur)    & Øxff)
#define TO(row,col)     vb_setcur(((row)<<8) | ((col) & Øxff))
#define BELL()          vb_putchar(Øx07)
#define NEWLINE()       { vb_putchar('\r'); vb_putchar('\n'); }

/*------------------------------------------------------------*/
static char     *Bp;     /* Points at current cursor pos.    */
static char     *Maxbp;  /* Points at rightmost char on line*/
static char     *Startp; /* Points at start of buffer        */
static char     *Endp;   /* Points at end of buffer          */
static short    Home;    /* Home cursor position             */
static int      Scroll;  /* Number of lines scrolled         */
static int      Insert;  /* True if in insert mode           */

/*------------------------------------------------------------*/
static int getkey()
{
        /* Return a key from the keyboard. Keys are gotten in
         * raw input mode and mapped as specified above if
         * necessary.
         */

        register int    c;
        static   int    ateof   = Ø;

        if( ateof )
                return EOF;
        if( !(c = bdos(BDOS_IN) & Øxff) )
        {
                switch( bdos(BDOS_IN) & Øxff )
                {
                case _LEFT:      c = LEFT       ; break;
                case _RIGHT:     c = RIGHT      ; break;
                case _CTL_LEFT:  c = CTL_LEFT   ; break;
                case _CTL_RIGHT:c = CTL_RIGHT   ; break;
                case _INS:       c = INS        ; break;
                case _DEL:       c = DEL        ; break;
                case _HOME:      c = HOME       ; break;
                case _END:       c = END        ; break;
                case _UP:        c = UPKEY      ; break;
                case _DOWN:      c = DOWNKEY    ; break;
                default:         c = NULL       ; break;
                }
        }
        else if( c == '\r' )    /* map ENTER key to '\n' */
        {
                c = '\n' ;
        }
        else if( c == CNTL_C || c == CNTL_Z )
        {
                ateof = 1 ;
                c     = EOF;
        }
        return c;
}
/*------------------------------------------------------------*/
static  int     pchar(c,scroll_ok)
{
        /* Print a character by replacing the character at the
         * current cursor position and moving right. The screen
         * is not permitted to scroll unless scroll_ok is true.
         * 1 is returned if the screen scrolled
         */
        vb_replace(c);
        return right( scroll_ok );
}
```

```
/*------------------------------------------------------------*/
static void     ptail( bp, end, move )
register char   *bp, *end;
{
        /* Print out all chars between bp and end (inclusive)
         * without modifying the current cursor position. If
         * move is zero the cursor will not change position,
         * otherwise the cursor will be left pointing to the
         * character referenced by end.
         */

        register short  posn;

        if( !move )
                posn = vb_getcur();

        while( *bp && bp <= end )
                if( pchar(*bp++, 1) )
                {
                        UP(posn);       /* If scrolled update*/
                        UP(Home);       /* initial and Home  */
                        Scroll++;       /* cursor posn       */
                }

        if( !move )
                vb_setcur( posn ); /* cursor under first char*/
        else
                left();
}
/*------------------------------------------------------------*/
static int      addchar( c )
{
        /* If we aren't at the right-most extreme of the
         * buffer, move the tail over to make room for the
         * current character, else just print it. Ring the
         * bell if there's no more room in the buffer.
         * Return 1 if we should advance bp (ie. didn't
         * reach eos) else return Ø.
         */

        register int rval   = Ø;

        if( Bp <= Endp  &&  (' ' <= c  &&  c < Øxff) )
        {
                if( Insert && Bp <= Maxbp )
                {
                        /* Move everthing to the right over
                         * one notch
                         */

                        if( Maxbp < Endp )
                                Maxbp++ ;

                        memmove( Bp+1, Bp, Endp-Bp );
                        ptail( Bp, Maxbp, Ø );
                }

                *Bp = c ;       /* Put it in the buffer */

                if( Bp >= Endp )
                {
                        vb_replace( c );
                        BELL();
                }
                else
                {
                        if( pchar(c, 1) ) /* We've scrolled.*/
                        {  /* Adjust the home cursor posn.  */
                                UP( Home );
                                Scroll++;
                        }

                        rval++;
                        if( Bp+1 > Maxbp )
                                Maxbp = Bp+1 ;
                }
        }
        return( rval );
}
```

```
/*-----------------------------------------------------------*/
static int left()
{
        register short  row,col;

        if( Bp <= Startp )
        {
                BELL();
                return 0;
        }
        else
        {
                row = vb_getcur();
                col = COL( row );
                row = ROW( row );

                if( col == 0 )
                        TO( row-1, 79 );
                else
                        TO( row, col-1 );

                --Bp;
                return 1;
        }
}

/*-----------------------------------------------------------*/
static void left_word()
{
        if( Bp <= Startp )
                BELL();
        else
        {
                do {
                        left();
                } while( Bp > Startp && *Bp == ' ' );

                while(  Bp > Startp && *Bp != ' ' )
                {
                        left();
                }

                if( *Bp == ' ' )
                {
                        right(0);
                        Bp++;
                }
        }
}

/*-----------------------------------------------------------*/
static int right( scroll_ok )
{
        register short  row, col;

        /* Move the cursor right. If scroll_ok is true the
         * screen will scroll when the cursor moves past the
         * end of the bottom line 1 is returned if the screen
         * scrolled, 0 otherwise.
         */
        if( Bp >= Endp )
                BELL();
        else
        {
                col  = vb_getcur();
                row  = ROW( col );
                col  = COL( col );

                if( col != 79 )
                        TO( row,   col+1 );
                else if( row != 24 )
                        TO( row+1, 0     );
                else if( scroll_ok )
                {
                        NEWLINE();
                        return 1;
                }
        }
        return 0;
}

/*-----------------------------------------------------------*/
static void right_word()
{
        for(; Bp < Maxbp  &&  *Bp != ' ' ; Bp++ )
                right(0);
        for(; Bp < Maxbp  &&  *Bp == ' ' ; Bp++ )
                right(0);
}

/*-----------------------------------------------------------*/
static void tab()
{
        do
        {       if( addchar(' ') )
                        Bp++ ;
                else
                        break ;
        }
        while( (Bp - Startp) & 0x7 );
}

/*-----------------------------------------------------------*/
static void delete()
{
        if( Bp >= Maxbp )
                vb_replace( *Maxbp = ' ' );
        else
        {
                memmove( Bp, Bp+1, Maxbp-Bp );
                *Maxbp = ' ';
                ptail( Bp, Maxbp-- , 0);
        }
}

/*-----------------------------------------------------------*/
static void era_left()
{
        if( left() )
                delete();
}

/*-----------------------------------------------------------*/
static void go_home()
{
        Bp = Startp ;
        vb_setcur( Home );
}

/*-----------------------------------------------------------*/
static void era_line()
{
        register char   *p;

        vb_setcur( Home );
        for( p = Startp; p < Maxbp; *p++ = ' ')
                pchar(' ', 0);
        vb_setcur( Home );
        Maxbp = Bp = Startp;
}

/*-----------------------------------------------------------*/
static void go_end()
{
        register int    i;
        if( Bp < Maxbp )
        {
                for( i = Maxbp-Bp; --i >= 0 ; right(0) )
                        ;
                Bp = Maxbp;
        }
}

/*-----------------------------------------------------------*/
static void change_insertmode()
{
        if( Insert = !Insert )
                vb_normalcur();
        else
                vb_blockcur();
}
```

```
/*-----------------------------------------------------------*/
void init_egets( buf, bufsize )
char    *buf;
{
        Startp   = buf;
        Endp     = Startp + (bufsize-1) ;
        *Endp--  = '\0' ;
        Bp       = Startp;
        Maxbp    = Startp;
        Home     = vb_getcur();
        Scroll   = 1;
        Insert   = 1;

        /* If the buffer's empty fill it with spaces, otherwise
         * use the existing contents but fill the remainder
         * with spaces.
         */

        if( *Bp )          /* If the buffer isnt empty print out*/
        {                  /* its contents and set maxbp to     */
                while( *Bp )    /* point at the previous end */
                    if( pchar( *Bp++ ,1) )     /* of string.*/
                    {                      /* Update Home if    */
                        UP( Home );        /* the screen scrolls*/
                        Scroll++;
                    }
                Maxbp = Bp ;
                vb_setcur( Home );         /* Home cursor */
        }

        for( ; Bp <= Endp ; *Bp++ = ' ' ) /* and then fill */
                ;                /* the rest of it with spaces. */
        Bp = Startp;
}

/*-----------------------------------------------------------*/
char    *egets( buf, bufsize )
char    *buf;
{
        /* Get a string with editing. If bufsize is wider than
         * your screen, strange things will happen when you try
         * to use the editing functions. If you access this
         * function via fgetline() then it will input longer
         * lines in 78 character chunks and \<CR> can be used
         * to extend a line.
         *
         * ^H or BACKSPACE Destructive backspace, close up
         *                 remainder of string to fill hole.
         * LEFT   CURSOR   Non-destructive backspace
         * RIGHT  CURSOR   Move right one character.
         * ^LEFT  CURSOR   Left  to previous word or line start
         * ^RIGHT CURSOR   Right to next word or line end
         * HOME            Left edge of line
         * END             Right edge of line
         * CR or LF        Terminate line.
         * ^X              Erase entire line but don't return.
         * ESC             Return a null string immediately.
         * INS             Toggle from insert to overwrite mode
         * DEL             Delete a current cursor position and
         *                 close up to fill hole.
         * HT              ^I, expanded to equivalent number
         *                 of spaces.
         * Any character   Enter that character at cursor posn
         * Anything else   Ring the bell.
         *
         * The bell will also ring if you try to move the
         * cursor past either the left or right edges of the
         * buffer. Return a pointer to the end of string
         * normally, return 0 on EOF.
         */

        register int    c ;       /* Current character. */

        init_egets( buf, bufsize );

        /* Bp points into the bufer at the current cursor
         * location. end points at the righmost place that the
         * cursor movement commands can get us. There is
         * actually one more place in the buffer. Get the line:
         */

        while(  (c = getkey()) != '\n'  &&  c != EOF )
        {
                switch( c )
                {
                case LEFT:       left        ( );             break;
                case RIGHT:      right       (0);     Bp++;   break;
                case CTL_LEFT:   left_word   ( );             break;
                case CTL_RIGHT:  right_word( );               break;
                case '\t':       tab         ( );             break;
                case HOME:       go_home     ( );             break;
                case END:        go_end      ( );             break;
                case '\b':       era_left    ( );             break;
                case DEL:        delete      ( );             break;
                case CAN:        era_line    ( );             break;
                case INS:        change_insertmode();         break;
                case UPKEY:
                case DOWNKEY:    /* ignore */                 break;
                default:         Bp += addchar(c);            break;

                case ESC:        *Startp = 0;
                                 NEWLINE();
                                 return( Startp );
                }
        }

        /* Delete trailing whitespace, terminate the string, go
         * to the next line, and return EOF if we're at end of
         * file, the end pointer otherwise.
         */

        for( ; *Endp == ' ' && Endp >= Startp ; --Endp )
                ;

        *++Endp = '\0';
        go_home();                  /* Home the cursor and then  */
        while( --Scroll >= 0 )      /* output enough blank lines */
                NEWLINE();          /* to get past the current   */
                                    /* input line                */

        vb_normalcur();             /* Restore the normal cursor */

        return ( c == EOF  &&  Startp == Endp ) ? NULL : Endp ;
}

/*-----------------------------------------------------------*/
char    *getl( buf, maxline, fp )
char    *buf;
FILE    *fp;
{
        /* Works exactly like fgets but returns a pointer to
         * the end of the string on success and doesn't put
         * the newline in the string. NULL is returned on EOF
         * or on the first character in the buffer being ^Z or
         * if the buffer is empty.
         */

        register int    c;
        register char   *bp = buf;
        int             sawslash = 0;

        while( (c = fgetc(fp)) != EOF  &&  --maxline > 0 )
        {
                if( c == CTL_Z )
                        return NULL;

                if( c == '\n' )
                {
                        if( !sawslash )
                                break;
                        --bp;             /* Delete the \     */
                        continue;         /* and ignore the \n */
                }

                *bp++ = c;
                sawslash = (c == '\\');
        }

        *bp = '\0' ;
        return( c == EOF && bp == buf ? NULL : bp );
}
```

```
/*-----------------------------------------------------------*/
char    *efgets( buf, maxline, fp )
char    *buf ;
FILE    *fp  ;
{       /*      An editing version of fgets. /*
        return (fp == stdin) ? egets( buf, maxline )
                             : getl ( buf, maxline, fp );
}

/*-----------------------------------------------------------*/
#ifdef DEBUG
testfile()
{
        static char buf[80];
        register FILE   *fp;
        if( !(fp=fopen("foo","r")) )
                printf("can't open foo\n");
        else
        {
                while( efgets(buf, 80, fp) > 0  )
                        printf("%s\n", buf );
        }
}

test256()
{
        static char buf[256] =
        "123456789 123456789 123456789 123456789 123456789 "
        "123456789 123456789 123456789 123456789 123456789 "
        "123456789 123456789 123456789 123456789";

        printf("line: ");
        while( efgets(buf, 256, stdin) > 0  &&  *buf )
        {
                printf("----->%s<---\n\nline: ", buf );
                *buf = 0;
        }
}
```

```
test40()
{
     static char buf[80] = "|         |         |         |         |";
     printf("               1         2         3         4\n");
     printf("      1234567890123456789012345678901234567890\n");
     printf("      ");
     while( efgets(buf, 40, stdin)  >  0  &&  *buf )
     {
     printf("      %s<---\n\n", buf );
     printf("               1         2         3         4\n");
     printf("      1234567890123456789012345678901234567890\n");
     printf("      ");
     *buf = 0;
     }
}

main()
{
        char buf[80];
        for(;;)
        {
                printf("\n0 - 40  column test\n");
                printf("1 - 256 column test\n");
                printf("2 - file test\n");
                printf("select -> ");
                gets( buf );
                switch( *buf )
                {
                case '0': test40();     break;
                case '1': test256();    break;
                case '2': testfile();   break;
                default:  exit(0);
                }
        }
}
#endif
```

Listing 2: *EFGETS.C für die Bildschirmeditierung.*

Frage & Antwort

Sprachschnittstellen

F: *Ich habe unter Microsoft C 4.0 zahlreiche Routinen entwickelt, die ich jetzt gerne im Microsoft QuickBASIC weiterverwenden möchte. Wie kann ich dies bewerkstelligen?*

A: Microsoft QuickBASIC in der Version 4.0 läßt sich sehr einfach mit Ihren C-Routinen kombinieren. Sie müssen einige Konventionen beachten, aber Sie können dies auf eine von zwei Möglichkeiten durchführen.

Sie können Ihre QuickBASIC-Programme von der MS-DOS-Kommandozeile aus mit BC (Microsoft QuickBASIC Kommandozeilen-Compiler) übersetzen und dann die entstandene Objektdatei mit Hilfe von LINK mit jeder C-Objektdatei oder Bibliothek linken. Tatsächlich können die meisten der C-Bibliotheksfunktionen aufgerufen werden. Sie können aber auch Ihre C-Routinen kompilieren, in eine Quick-Bibliothek binden und diese in die Microsoft Quick-BASIC-Umgebung laden. Die Routinen sind dann innerhalb des Environments verfügbar.

Ohne alle Details auflisten zu wollen, sind doch einige Dinge zu beachten. Jede von Microsoft QuickBASIC aufgerufene C-Funktion muß entweder als `SUB` oder als `FUNCTION` deklariert werden (*Listing 1*). Eine `FUNCTION` kann einen Wert zurück geben, eine `SUB`-Funktion aber nicht.

Des weiteren muß eine einheitliche Aufruffolge vereinbart werden. Das Schlüsselwort `CDECL` in der `DECLARE`-Anweisung zum Beispiel stellt sicher, daß die C-Konvention zur Paramterübergabe eingehalten wird. Die Reihenfolge geht in umgekehrter Richtung (von rechts nach links), so wie es die C-Routine erwartet. Das ist der am häufigsten angewendete Fall, der keine Änderung Ihrer C-Module erfordert. Es ist aber auch möglich, die Aufrufkonvention der aufgerufenen Funktion zu ändern. Die aufgerufene C-Funktion kann sich an BASIC angleichen, indem das Schlüsselwort `fortran` verwendet wird (*Listing 2*).

Drittens muß eine gemeinsame Namenskonvention eingehalten werden. Der C-Compiler erkennt 31 Zeichen lange Bezeichner und ändert Groß-/Kleinschreibung nicht. Des weiteren fügt er vor jede Routine einen Unterstrich. Die Funktion `called_proc` in Listing 2 wird zu `_called_proc`. BASIC erkennt bis zu 40 Zeichen, konvertiert alle Zeichen in Großschreibung und erlaubt Punkte in den Namen.

Das Schlüsselwort `CDECL` stellt sicher, daß die C-Namensgebungskonvention eingehalten wird. Punkte werden zu Unterstrichen und ein führender Unterstrich wird zur Variablen hinzugefügt. Da der Linker Groß-/Kleinschreibung ignoriert (wenn /NOI angegeben wird) wird `CALLED.PROC` zu `_Called_Proc`, und der Linker kann den externen Verweis lösen. Das bedeutet, daß die C-Routinen Groß-/Kleinschreibung nicht unterscheiden sollen - (wie bei `_called_proc` und `_CALLED_proc`), um verschiedene Funktionen aufzurufen.

```
DECLARE SUB Exchange CDECL (a%, b%)
' Es gibt kein Funktionsergebnis

DECLARE FUNCTION Power% CDECL (a%)
' Die Deklaration von Power% erlaubt
' die Rückgabe eines Funktionsergebnisses
```

Listing 1.

Bedenken Sie auch, daß `CDECL` es C nicht erlaubt, mehr als 31 Zeichen zu unterscheiden. Es gibt aber zwei Auswege. Der erste ist einfach das Vermeiden von Namen, die länger als 31 Zeichen sind. Der zweite ist die Verwendung des QuickBASIC-Schlüsselworts `ALIAS`, welches die Ersetzung von Namen erlaubt.

Viertens muß eine Konvention für die Parameterübergabe (die Daten selbst) vereinbart werden. Die Compiler von Microsoft übergeben Parameter auf dreierlei Art: »by value«, »by near reference« (nur Offset) und »by far reference« (Segment und Offset). Die aufrufende und die aufgerufene Funktion müssen dabei übereinstimmen. Eine »by reference« übergebene Variable gibt der aufgerufenen Funktion direkten Zugang zu der Variablen, so daß die Routine die Variable direkt ändern kann. Jede solche Änderung muß dann von der aufrufenden Routine in Betracht gezogen werden. Bei der Übergabe »by value« hat die aufgerufene Funktion keinen direkten Zugriff auf die Variable. Der Wert der Variablen ist bekannt, kann aber nicht geändert werden.

In BASIC wird die Übergabe »by near reference« ausgeführt. In C wird dies, außer bei Arrays, »by value« durchgeführt, außer bei der Verwendung von Zeigern (*Listing 3*). Die C-Funktion deklariert ihre drei Parameter als `near pointer`, was bedeutet, daß die Variablen »by near reference« übergeben werden. Da dies in BASIC voreingestellt ist, sind keine zusätzlichen Schlüsselwörter notwendig.

Listing 4 zeigt eine C-Funktion, die einen Parameter als »far reference«, und einen »by value« erwartet. Sehen Sie die `DECLARE`-Anweisung im Microsoft QuickBASIC-Programm. Das Schlüsselwort `SEG` bewirkt die Übergabe einer »far reference«. `BYVAL` erledigt die Übergabe »by value«.

```
int fortran called_proc (int a, char b)
{
/* Hier ist die eigentliche Funktion.
 * Beachten Sie das Schlüsselwort
 * fortran, das die Parameter entsprechend
 * der BASIC-Konvention übergibt
 */
}
```

Listing 2.

```
DECLARE SUB Exchange CDECL (a%, b%)

A% = 10
B% = 20
C% = 30


PRINT A%; B%; C%
CALL Exchange(A%,B%,C%)
PRINT A%; B%; C%
```

```
/* C-Routine exchange() */

void exchange(int near *x,
              int near *y, int near *z)
{
    int a1;
    int a2;

    a1 = *x;
    a2 = *y;
    *x = *z;
    *y = a1;
    *z = a2;
}

/* A%, B% und C% werden by NEAR
 * Reference übergeben um sie zu
 * tauschen. Die neuen Werte sind sofort
 * verfügbar, nachdem die C-Routine
 * beendet wird.
 */
```

Listing 3.

Dies steht im Gegensatz zu der Standardübergabe in BASIC. `CALL CALC((Q))` übergibt eine Referenz, aber die Referenz bezieht sich auf eine temporär erzeugte Variable, nicht auf die Originalvariable. Der Wert der Originalvariablen wird somit bewahrt. `BYVAL` erlaubt die Übergabe eines Wertes, anstatt der Referenz der temporären Variablen.

Arrays sollten immer mit einer »far reference« durch die Verwendung des Schlüsselworts `SEG` übergeben werden. Arrays von dynamischen Strings und auch `$STATIC`-Arrays sind `Near`-Referenzen, aber alle anderen Arrays Far-Referenzen. Wenn Sie sich nicht ganz sicher sind, ob eine `Near`-Referenz mit dem Array funktioniert, verwenden Sie das Schlüsselwort `SEG`, das immer richtig ist.

```
DECLARE FUNCTION AddValue CDECL (SEG a%,
BYVAL b%)

D% = 10
E% = 20
F% = D% + E%
PRINT D%; E%; F%
PRINT D%; E% AddValue(D%, E%)

' Die Deklaration der FUNCTION AddValue
' erlaubt die Rückgabe eines
' Funktionsergebnisses
```

```
/* C-Routine addvalue() */

int addvalue(far *y, int n)
{
    int sum;

    sum = *y + n;

    return(sum);
}


/* Die C-Funktion erwartet eine FAR
 * Referenz für den ersten Wert, wie an
 * SEG a% zu sehen ist, und erhält
 * ihren zweiten Wert by Value, wie an
 * BYVAL b% zu sehen ist.
 */
```

Listing 4.

Zu guter Letzt muß auf die Speichermodelle geachtet werden. Microsoft QuickBASIC verwendet ausschließlich Far-Codeadressen. Das bedeutet, daß alle C-Routinen, die mit QuickBASIC verwendet werden, im `medium-`, `large-` oder `huge`-Speichermodell übersetzt werden müssen. Dies ist der einfachste Weg. Das `Small-` oder das `Compact`-Speichermodell kann verwendet werden, wenn die Funktion mit dem Zusatz `FAR` versehen wird.

Es gibt andere Dinge zu beachten, aber diese hier sind die grundlegenden. Wenn Sie aber einmal die Regeln für die gemischte Programmierung in verschiedenen Sprachen verstanden haben, können Sie nicht nur C und BASIC mischen, sondern auch alle anderen.

Programme testen mit CodeView:

Debuggen leicht gemacht

Das Debuggen ist eine so schreckliche, langweilige, zeitaufwenige Tätigkeit, daß niemand viel Gedanken an sie verschwenden möchte. Wahrscheinlich hinken aus diesem Grund gute Debugger in der Entwicklung immer den Compilern hinterher. Erst mit der Einführung von SYMDEB (SYMbolischer DEBugger, zuerst mit dem Makro-Assembler 3.0 ausgeliefert) bot Microsoft den Programmierern eine Alternative zum unzureichenden DEBUG an.

Der CodeView-Debugger, der zuerst mit dem Microsoft C-Compiler Version 4.0 ausgeliefert wurde, vertritt eine neue Generation. Ähnlich wie SYMDEB gegenüber DEBUG neue Möglichkeiten durch das Einlesen der Quelldateien sowie Anzeigen der symbolischen Referenzen der globalen Variablen und Unterprogramme bietet, verbessert CodeView die Möglichkeiten gegenüber SYMDEB durch die Anzeige von lokalen Variablen und Unterprogrammen, neue Datenanzeige- und Eingabetypen, die 8087-Coprozessor-Registeranzeige sowie weitere Debugging-Techniken.

CodeView verwendet eine Ganzseitenanzeige mit Fenstern und bei der Benutzerschnittstelle Maus, Menü und Kommandozeilen; Codeview kann auch C-Ausdrucksauswertung durchführen. Derzeit ist CodeView mit den Microsoft-Programmiersprachen C, Pascal, FORTRAN, Assembler und BASIC einsetzbar. Nach einem Überblick über seine Fähigkeiten werden in diesem Artikel Details anhand einer Debug-Sitzung demonstriert.

Bild 1 zeigt eine Bildschirmanzeige von CodeView. Zur Informationsanzeige sind vier Fenster vorhanden: das Quelltextfenster, das Dialogfenster, das Registerfenster und das Watchfenster.

Den Mittelpunkt bildet das Quelltextfenster, das man sich auch als Nur-Lese-Editor mit Debug-Möglichkeiten vorstellen kann. Der Benutzer kann durch den Quelltext blättern oder nach bestimmten Teilen suchen. Die aktuelle Zeile und die aktiven Breakpoints werden durch einen blauen Hintergrundbalken sowie intensive Textdarstellung angezeigt. Mit einigen Kommandos ist das Durchsehen eines Programms, das auch aus mehreren Dateien bestehen kann, möglich. Der Anwender kann ein Programmlabel suchen, und in das Quellfenster wird automatisch die zugehörige Datei geladen und angezeigt.

Im Dialogfenster sind Anzeigen untergebracht, die nicht in ein einheitliches Fensterschema passen wie die Datenausgabe für unterschiedliche Datentypen. Das Dialogfenster ist die Kommandozeilenschnittstelle von CodeView. Tatsächlich verhält sich CodeView, wenn er mit der Option `/T` aufgerufen wird, wie sein Vorgänger SYMDEB in seiner nicht bildschirm- und fensterorientierten Betriebsweise. Das Dialogfenster zeigt nur einen Teil des Dialogtextes. Scrollkommandos, ähnlich denen im Quellfenster, können zum Anschauen der Daten benutzt werden.

```
File View Search Run Watch Options Language Calls Help | F8=Trace F5=Go
                                 where.c
0) envptr,s  :  "c:\msc5\bin;d:\bin\dos"                    AX = 1154
1) ptr,s   :  "command"                                     BX = 008A
                                                            CX = 0000
6B21:0092 8946B8        MOV      Word Ptr [found],AX        DX = 11DD
6B21:0095 3D0000        CMP      AX,0000                    SP = 114E
6B21:0098 7403          JZ        main+8d (009D)            BP = 11A8
6B21:009A E97D00        JMP      011A                       SI = 008A
42:                            envpath = envptr;            DI = 11DD
6B21:009D 8B46AA        MOV      AX,Word Ptr [envptr]       DS = 6D5C
6B21:00A0 8946FE        MOV      Word Ptr [envpath],AX      ES = 6D5C
43:                            while(*envpath) {            SS = 6D5C
6B21:00A3 8B5EFE        MOV      BX,Word Ptr [envpath]      CS = 6B21
6B21:00A6 803F00        CMP      Byte Ptr [BX],00           IP = 0079
6B21:00A9 7503          JNZ       main+9e (00AE)
6B21:00AB E96C00        JMP      011A                        NV UP
44:                                 ptr = pathbuf;           EI PL
6B21:00AE 8D46BA        LEA      AX,Word Ptr [pathbuf]       NZ NA
6B21:00B1 8946FC        MOV      Word Ptr [ptr],AX           PE NC

>w?ptr,s
>?argc
2
>
```

Bild 1: *Dieses Bild zeigt den CodeView-Bildschirm mit dem Quelltextfenster, dem Dialogfenster, dem Registerfenster und dem Watchfenster.*

Das Registerfenster zeigt die Register des Prozessors an. Das einzige zur Verfügung stehende Kommando ist die Möglichkeit, Flags durch Anklicken mit der Maus zu ändern.

Das Watchfenster dient zum ständigen Betrachten von Variablen und Daten, die für den Benutzer interessant sind, so daß Anzeige-Kommandos nicht immer wieder eingegeben werden müssen. Das Anzeigeformat im Watchfenster stimmt mit dem im Dialogfenster überein. Doch die Kommandoeingabe für Variablen/Daten in diesen zwei Fenstern ist unterschiedlich. Beispielsweise zeigt `?i` die Variable `i` im Dialogfenster an, `w?i` hingegen zeigt sie im Watchfenster an.

Die Pull-Down-Menüs sind denen von Windows nachempfunden, einschließlich Maus- und Tastaturhandhabung.

Vorbereitungen für das Debuggen

Die Vorbereitung eines Programms für das Debuggen mit CodeView ist einfach. Beim Kompilieren müssen Sie nur den Schalter `/Zi` verwenden:

```
CL -c -Zi Programm
```

Diese Option weist den Compiler an, die Namen, Adressen und Typen der globalen und lokalen Variablen, die Namen, Adressen und Rückgabewerte der globalen und statischen Funktionen und die Quelldatei-Zeilennummern mit in die Objektdatei zu schreiben.

Wenn Sie diese Informationen nicht benötigen (wenn das Modul schon ausgetestet ist), können Sie auch angeben, daß nur globale Funktionen und Variablen sowie Zeileninformationen in die Objektdatei gelangen, indem Sie die Option `/Zd` beim Kompilieren benutzen:

```
CL -c -Zd Programm
```

Dieser Schalter ist derselbe, der beim Debuggen mit SYMDEB benötigt wird.

Wahrscheinlich werden Sie auch die Optimierung des Compilers durch die Angabe des Schalters `/Od` abschalten wollen. Hierdurch ist die Übereinstimmung von Quelldatei und dem erzeugten Code größer.

Beim Linken müssen Sie die Option `/CO` (CodeView) verwenden.

```
LINK Programm /CO;
```

Der Schalter `/CO` weist den Linker an, die symbolischen Informationen und Angaben über den Zugriffspfad der Objektdateien mit in die EXE-Datei zu übernehmen. CodeView benutzt die Pfadangabe, um die C-Quelldatei zu finden. Sie sollten daher Quell- und Objektdateien im selben Unterverzeichnis speichern.

Sowohl CodeView als auch SYMDEB lassen sich bei der Einhaltung einiger Programmierregeln leichter handhaben. Da SYMDEB nur globale Variablen und Funktionen handhaben kann, werden Sie wahrscheinlich lokale Variablen und Funktionen seltener als statisch deklarieren. Bei CodeView brauchen Sie allerdings keine solchen Änderungen vornehmen.

Die Programmierregeln bei der Verwendung von CodeView sind dieselben, wie bei der Erstellung von strukturierten, gut lesbaren C-Programmen. Sie sollten kleine Funktionen programmieren. Sie sollten nur ein Statement in eine Zeile schreiben, da mehrzeilige Quellcode-Statements in der gemischten Quell- und Objektdarstellung von CodeView etwas sonderbar aussehen. Sie erhalten eine bessere Vorstellung vom Zusammenhang von Quellcode und kompiliertem Code, wenn Sie lange, komplexe Anweisungen vermeiden.

Andererseits zeigt CodeView nicht den Quellcode von Include-Dateien oder Makros. Ebenso kann CodeView keine Quellcodedarstellung von Funktionen aus den Bibliotheken zeigen, da der Librarymanager LIB die symbolischen Informationen entfernt, sollten sie auch mit der Option `/Zi` übersetzt worden sein. Es empfiehlt sich deshalb die Module erst nach dem vollständigen Austesten in die Bibliotheken zu übernehmen.

Hilfe durch MAKE

Die symbolischen Informationen, die der Linker für CodeView an die EXE-Datei anfügt, machen diese Datei sehr viel länger als normal. Auch werden Sie Programme mit symbolischen Informationen nicht vertreiben wollen, da sonst jeder beim Betrachten der Datei mit CodeView etwas über das Innenleben erfahren würde.

Keyboard/Mouse

Description	Keyboard	Mouse
Enter Help system	F1	Use Help menu
Open register window	F2	Use Options menu
Toggle display mode	F3	Use View menu
Switch to Output screen	F4	Use View menu
Go	F5	Click left on Go
Switch cursor window	F6	None
Execute cursor line	F7 at location	Click right on source line
Trace through routine	F8	Click left on Trace
Set breakpoint at cursor	F9 at location	Click left on source line
Step over routine	F10	Click right on Trace
Make window grow	CONTROL+G	Drag line up or down
Make window tiny	CONTROL+T	Drag line up or down
Scroll up a line	Move cursor off top	Click left on up arrow
Scroll up a page	PGUP	Click above elevator
Scroll down a line	Move cursor off bottom	Click left on down arrow
Scroll down a page	PGDN	Click below elevator

Run Commands

Trace	T [count] (F8) (Click left on Trace)	Executes current source line or instruction. If count is given, repeats count times. Traces through routines and interrupts.
Program Step	P [count] (F10) (Click right on Trace)	Executes current source line or instruction. If count is given, repeats count times. Steps over routines and interrupts.
Go	G [address] (F5) (Click on Go)	Executes until debugger encounters address, a previously set breakpoint, or the end of the program.
Execute	E	Executes in slow motion until you press a key.

Display Commands

Display Expression

? expression[,format] — Evaluates expression and displays the value. If format is given, the value is displayed in the corresponding format, as shown below:

Specifier	Input Type	Output Format
d or i	Integer	Signed decimal integer
u	Integer	Unsigned decimal integer
o	Integer	Unsigned octal integer
x or X	Integer	Hexadecimal integer
f	Real	Floating point
e or E	Real	Scientific notation
g or G	Real	"E" or "f," whichever is more compact
c	Character	ASCII equivalent of character
s	String	C null-terminated string

Integers can have l or s prefix.

Display Symbols

X[?[module!][routine.]symbol¦*] — Displays symbols as described below:

X?module!routine.symbol	This symbol in routine in module.
X?module!routine.*	All symbols in routine in module.
X?module!symbol	This symbol in module (must be static).
X?module!*	All static symbols in module. Will not find local or dynamic variables.
X?routine.symbol	This symbol in routine. Searches all modules.
X?routine.*	All symbols in routine. Searches all modules.
X?symbol	Looks for symbol in this order: current routine, current module, all modules.
X?*	All symbols in current routine.
X*	All module names.
X	All symbols in all modules.

Set Mode	S[+¦-¦&]	Sets display mode to source (+), assembly (-), or mixed (&).
Current Location	.	Puts current line in center of display window.
Stack Trace	K (ALT+C)	Displays routines and their arguments. The current routine is on top; the initial routine is on bottom.
Unassemble	U [range]	Displays instructions in range. If no range is given, scrolls down one screen.
View	V[expression] V[.[file:]line]	Displays source lines, starting at expression or line. If file is given, loads source file. If no arguments are given, scrolls down one screen.

Expressions

CodeView Expressions

Expressions in CodeView commands consist of symbols, constants, and/or operators. Operators join symbols and constants into larger expressions. For example, "+" joins the constants "4" and "5" in the expression "4+5". Syntax for operators and constants is presented in the next three screens.

Changing Expression Evaluators

```
USE [language]        Changes evaluator to AUTO, C, FORTRAN, or BASIC. If
                      language is not given, displays current evaluator.
```

You can also choose evaluators with the Language menu. AUTO uses source-file extension to select evaluator (with C as default choice).

Notes on Extended Operators (. and :)

```
[routine.]variable     Local or global variable
.[module:]linenumber   Source-line address
```

C Expressions

```
Precedence   Operators
highest  1 | ()  []  ->  .
         2 | !  ~  -  (type)  ++  --  *  &  sizeof
         3 | *  /  %  :
         4 | +  -
         5 | <  >  <=  >=
         6 | ==  !=
         7 | &&
         8 | ||
         9 | =  +=  -=  *=  /=  %=
lowest  10 | BY  WO  DW
```

```
Syntax      Radix
Onumber   | Octal
Onnumber  | Decimal
Oxnumber  | Hexadecimal
```

```
Memory Operators
BY n   Return byte at n
WO n   Return word at n
DW n   Return dbl. word
```

FORTRAN Expressions

```
Precedence      Operators
highest 1 | ()
        2 | .  :
        3 | +  -  (unary)
        4 | *  /
        5 | +  -  (binary)
        6 | .LT. .LE. .EQ. .NE. .GT. .GE.
        7 | .NOT.
        8 | .AND.
        9 | .OR.
       10 | .EQV. .NEQV.
lowest 11 | =
```

```
Constant        Radix
number        | Default radix
radix#number  | Specified radix
#number       | Hexadecimal
```

BASIC Expressions

```
Precedence      Operators
highest 1 | ()
        2 | .  :
        3 | +  -  (unary)
        4 | *  /
        5 | \  MOD
        6 | +  -  (binary)
        7 | =  <>  <  >  <=  >=
        8 | NOT
        9 | AND
       10 | OR
       11 | XOR
       12 | EQV
       13 | IMP
lowest 14 | LET variable =
```

```
Constant     Radix
number     | Default radix
&Onumber   | Octal
&number    | Octal
&Hnumber   | Hexadecimal
```

```
Suffix   Type
  %    | Integer
  &    | Long
  !    | Single
  #    | Double
```

Watch/Break

```
Watch          W? expression[,format]    Displays expression in format,
Expression     W[type] range             or range in type format.

Watchpoint     WP? expression            Breaks execution when expression
                                         is true (non-zero).

Tracepoint     TP? expression[,format]   Breaks when expression or any
               TP[type] range            value in range changes. Displays
                                         same as Watch Expression.

Watch List     W                         Lists watch statements.

Watch Delete   Y number|*                Deletes indicated watch
                                         statement, or all if * is given.

Breakpoint  BP addr [count] ["cmnd"]  Sets breakpoint at address addr. If
Set                                   count is given, breakpoint is taken
                                      after count times. If cmnd is given,
                                      the command is executed at each break.

Breakpoint  BC list | *               Deletes breakpoints in list, or all
Clear                                 breakpoints if * is given instead.

Breakpoint  BD list | *               Disables breakpoints in list, or all
Disable                               breakpoints if * is given instead.

Breakpoint  BE list | *               Enables breakpoints in list, or all
Enable                                breakpoints if * is given instead.

Breakpoint  BL                        List breakpoints, including status,
List                                  location, pass count, and associated
                                      commands.
```

Memory Operations

Memory Type Formats--used with Dump, Watch, and Enter commands:

```
A | ASCII-format string
B | Byte (8-bit hexadecimal unsigned integer)
I | Integer (16-bit decimal)
U | Unsigned (16-bit decimal unsigned integer)
W | Word (16-bit unsigned hexadecimal)
D | Double Word (32-bit unsigned hexadecimal)
S | Short Real (4 bytes)
L | Long Real (8 bytes)
T | Ten-byte Real
```

Specifying Addresses and Address Ranges

```
expression1 [:expression2]   Specifies offset or segmented address.
address1 address2            Range from address1 to address2.
address1 L size              Range beginning at address1 and of length
                             size.

Enter      E[type] addr [list]    Enters values of type, beginning at addr.
                                  Debugger will prompt if list is not given.
                                  Items in list are separated by spaces.

Dump       D[type] [range]        Dumps values of type in range. Previous or
                                  default type assumed if type is not given.

Assemble   A [address]            Assembles mnemonic instructions beginning
                                  at address. Assumes next instruction if
                                  address is not given.

Register   R[reg [expression]]    Sets one of the following registers: AX, BX,
                                  CX, DX, SP, BP, SI, DI, DS, ES, SS, CS, IP,
                                  or 386 extended register.

           RF [flag]              Sets a condition from one of these pairs:
                                  OV-NV, DN-UP, EI-DI, NG-PL, ZR-NZ, AC-NA,
                                  PE-PO, CY-NC.

Search Memory    S range list       Displays each address in range that
                                    begins a sequence matching each byte
                                    in list.

Compare Memory   C range address    Compares bytes in range to an equal-
                                    sized block beginning at address.

Fill Memory      F range list       Fills range with byte values in list,
                                    repeating values until range filled.

Move Memory      M range address    Copies values in range to an equal-
                                    sized block beginning at address.

Port Input       I port             Displays value at port.
Port Output      O port byte        Assigns byte to port.

8087             7                  Displays 8087 registers and stack.
```

System Commands

```
Help       H                  Enters CodeView Help System.

Load       L [arguments]      Reloads program. Uses new command-line
                              arguments if given.

Option     O[{F|B|C|3}[+|-]]  Sets (+), clears (-), or displays the
                              status of Flip/Swap, Bytes Coded, Case
                              Sense, or 386 option.

Quit       Q                  Exits the CodeView debugger.

Radix      N [radix]          Sets radix to 8, 10, or 16, if radix is
                              given; otherwise displays current radix.

Redraw     @                  Redraws display screen.

Screen Exchange  \             Displays the output screen. Press any key
                               to return to the debugging screen.

Shell Escape     ![command]    Starts a new DOS shell and executes the
                               program or DOS command given as command.
                               If command is not given, user can enter
                               commands from the DOS prompt, and then
                               enter EXIT when ready to return to the
                               DOS prompt.

Tab Set          #number       Sets the number of spaces per tab, to
                               number.
```

Redirection Commands

```
[T]>[>]device   Redirects command output to device (file or device). If
                T given, output is echoed on CodeView screen. If second
                > given, output is appended.

<device         Redirects command input from device (file or device).

=device         Redirects both command input and command output.
```

Commands Used with Redirection

```
*comment        Displays comment but does not attempt to execute.
"               Pauses until user presses a key.
:               Delays for half a second.
```

Tabelle 1: *Die Hilfebildschirme von CodeView geben einen guten Überblick über seine Leistungsfähigkeit.*

Sie können MAKE benutzen, um die Dateien je nach Verwendung mit oder ohne CodeView-Informationen zu übersetzen. Am Beginn der MAKE-Datei werden einige Makros definiert.

```
CVC = /Zi /Od
CVL = /CO
COMP = msc $* $(CVC);
LINK = link $** $(CVL);
```

CVC und CVL enthalten die CodeView-Schalter für das Kompilieren und Linken. Nach diesen Definitionen können Sie die MAKE-Datei ähnlich der folgenden erstellen.

```
programm.obj : programm.c
        $(COMP)

modul1.obj : modul1.c
        $(COMP)

modul2.obj : modul2.c
        $(COMP)

programm.exe : programm.obj\
        modul1.obj\
        modul2.obj
        $(LINK)
```

Beim Erstellen der endgültigen EXE-Datei ohne die symbolischen Informationen bedarf es nur eines Rückgängigmachens der Definitionen in der Kommandozeile.

```
MAKE programm /DCVC= /DCVL=
```

Sie können aber auch das Dienstprogramm EXEPACK benutzen, um die symbolischen Informationen aus der EXE-Datei zu entfernen.

Start der Debug-Sitzung

In den meisten Fällen können Sie CodeView durch folgenden Aufruf starten:

```
CV Programm
```

Einige Situationen (wie der Einsatz eines PC-Kompatiblen oder eines S/W-Bildschirms an einem Farbadapter) können jedoch die Verwendung von Kommandozeilenoptionen nötig machen. Diese Schalter müssen vor der Angabe des Dateinamens der zu untersuchenden Datei eingegeben werden. Die wichtigsten Schalter bestimmen, wie CodeView die Bildschirmanzeige handhabt. Da CodeView ein Debugger mit Ganzbildschirmanzeige ist, muß er den Bildschirm mit den Ausgaben des zu untersuchenden Programms teilen. Je nach der verwendeten Hardware haben Sie einige Möglichkeiten, dieses Problem zu lösen.

Bei Systemen mit einem Videoadapter benutzt CodeView entweder das »Video-Swapping« oder das »Video-Flipping«, um Debugger und Programmanzeige auseinanderzuhalten. Bei der Verwendung eines monochromen Adapters benutzt CodeView das »Swapping«. CodeView allokiert Speicher, um den Bildschirmspeicherinhalt zu retten. Er tauscht die Bildschirminhalte zwischen diesem Speicher und dem Bildschirmspeicher, wenn Sie zwischen CodeView und Ihrem Programm hin- und herwechseln.

Verwenden Sie in Ihrem System einen Farb- (CGA) oder EGA-Adapter, so wendet CodeView das »Flipping« an. CodeView verwendet verschiedene Bildschirmspeicherseiten für die beiden Anzeigen. Flipping ist erheblich schneller als Swapping.

Sie können allerdings das Flipping mit einem Monochromadapter nicht einsetzen, da der 4 KByte große Bildschirmspeicher nur eine Seite beinhaltet. Ebenso können Sie das Flipping bei CGA- oder EGA-Adaptern nicht verwenden, wenn das Programm mehrere Videoseiten benutzt oder eine Grafikausgabe vornimmt. In diesem Fall müssen Sie die voreingestellte Option durch den Schalter `/S` (Swapping) ändern.

Sowohl Flipping als auch Swapping beinhalten einige Probleme. Bei jedem Tracen schaltet CodeView auf die Programmausgabe um und anschließend wieder zurück. Möglicherweise bekommen Sie eher Kopfschmerzen, als daß Sie den Programmfehler finden. Wenn Sie ein Programm debuggen, das die Ports des Videoadapters direkt beeinflußt, dann kann CodeView den Zustand nicht mehr herstellen, wenn es die Kontrolle an das Programm übergibt. Aber es gibt keinen Ausweg, wenn Sie Programme mit nur einem Monitor debuggen.

Diese Probleme sind gelöst, wenn Sie einen zweiten Bildschirmadapter verwenden. Sie müssen dann beim Programmstart die Option `/2` angeben, und CodeView benutzt dann für seine Ausgaben den zweiten Adapter.

Wie bei SYMDEB können Sie aber CodeViews Ein- und Ausgaben auch über eine serielle Schnittstelle an ein Terminal umleiten. Sie büßen aber einige der Vorteile von CodeViews Ganzschirmanzeige ein.

Menüs und die Maus

Obwohl das Pull-Down-Menü-Interface und die Mausunterstützung den Gebrauch von CodeView einfacher gestalten, als bei vielen anderen Debuggern, können Neulinge doch verwirrt werden. Tatsächlich gibt es mehrere Benutzerschnittstellen - eine SYMDEB-ähnliche Kommandozeilen-Benutzeroberfläche, eine Windows-ähnliche, die Funktionstasten und die Maus. Jede Benutzeroberfläche hat Vorzüge und teilweise sind sie ähnlich. Sie können die Kommandos in unterschiedlicher Weise eingeben, und diese Flexibilität läßt CodeView am Anfang etwas umständlich erscheinen. Die Menü-, Maus- und Funktionstastenbedienung stellt die am meisten benötigten Kommandos mehrfach zur Verfügung. Es wird keine zusätzliche Funktionalität, sondern nur ein erleichterter Umgang erreicht.

```
/* WHERE zeigt, wo DOS Ihr ausführbares Programm findet */
/* CodeView-Demo: das Programm läuft so nicht korrekt!! */

#include    <stdio.h>
#include    <time.h>
#include    <sys\types.h>
#include    <sys\stat.h>

#define NUMEXTS 3
#define MAXFILELEN  12
#define MAXPATHLEN  65

struct  stat    file_stat;
char    *getenv(char *);
char    *strchr();


main (argc, argv)
int argc;
char    **argv;
{
    int     found;
    char    pathbuf[MAXPATHLEN];
    char    namebuf[MAXFILELEN];
    char    *ptr = pathbuf;
    char    *envpath;
    char    *envptr;

    if (argc < 2) {
        printf("Bitte WHERE Dateiname[.EXT] eingeben \n");
        exit(2);
    }

    if(!(envptr = getenv("PATH")))
        envptr = "";

    /* Schleife durch die angegebenen Dateinamen */
    while(ptr = *++argv) {
        strcpy(namebuf, ptr);

        /* Zuerst das aktuelle Verzeichnis versuchen */
        if(!(found = search_for_file(namebuf,
                    strchr(namebuf, '.')))) {
            /* Schleife durch die Unterverzeichnisse
               in der Umgebungsvariablen PATH */
            envpath = envptr;
            while(*envpath) {
                ptr = pathbuf;
                /* Pfad in den Puffer kopieren */
                while(*envpath && *envpath != ';')
                    *ptr++ = *envpath++;

                /* Pfadtrennung überspringen */
                while(*envpath == ';')
                    ++envpath;
                /* Kein \ bei der Suche im Stammverzeichnis
                   anfügen */
                if(*(ptr - 1) != '\\')
                    *ptr++ = '\\';
                *ptr = '\0';
                strcat(pathbuf, namebuf);
            }
        }
        if(search_for_file(pathbuf, strchr(namebuf, '.'))) {
            printstats(pathbuf);
            break;
        } else {
            printstats(namebuf);
        }
    }
}


search_for_file(fullpath, specific_file)
char    *fullpath;
int     specific_file;
{
    int     found, next_ext;
    static  char *extension[NUMEXTS] = {".COM",".EXE",".BAT"};

    if(specific_file)
        found = !stat(fullpath, &file_stat);
    else {
        for(next_ext = 0; next_ext < NUMEXTS; ++next_ext) {
            /* Erweiterung .COM, .EXE und .BAT versuchen */
            strcat(fullpath, extension[next_ext]);
            if(found = !stat(fullpath, &file_stat))
                break;
        }
    }

    return(found);
}


/* Ausgabe des ganzen Namens des übergebenen
   Strings und stats (Größe, Datum...)
   von der globalen Struktur file_stat
*/
printstats(fullname)
char    *fullname;
{
    struct  tm  *tmptr;

    tmptr = localtime(&file_stat.st_atime);
    printf("%s \t%ld\t%2d-%2d-%2d\t%d:%02d%c\n",
        strupr(fullname),
        file_stat.st_size,
        tmptr->tm_mon+1,
        tmptr->tm_mday,
        tmptr->tm_year,
        (tmptr->tm_hour<13 ?tmptr->tm_hour :tmptr->tm_hour-12),
        tmptr->tm_min,
        (tmptr->tm_hour <= 12 ? 'p' : 'a'));
}
```

Listing 1: *Das Listing der Utility WHERE.*

Trace, Go und Break

Schauen wir uns einige Möglichkeiten von CodeView an, um die Debugmöglichkeiten kennenzulernen. Die Anwender von DEBUG und SYMDEB sind mit den Trace- und Go-Kommandos vertraut. DEBUG Version 3.0 und SYMDEB beinhalten auch noch ein Proceed-Kommando. CodeView hat alle drei Möglichkeiten implementiert, um die Programmausführung zu überwachen.

Das Trace-Kommando benutzt die Einzelschritt-Trap-Anweisung des Intel Mikroprozessors, um das Programm befehlsweise auszuführen. Wie bei DEBUG oder SYMDEB können Sie mit CodeView eine Einzelbefehlsausführung durchführen, indem Sie im Dialogfenster `T` eingeben. Sind Sie im Assembler-Modus, führen Sie eine Assembleranweisung aus. Im Quellcode-Modus führen Sie eine Quellcodeanweisung aus. Sie können aber auch einen Einzelschritt durch Drücken der Funktionstaste F8 oder das Anklicken von `F8=Trace` mit dem linken Mausknopf ausführen. Das Ausführen mehrerer Befehle ist aber nur im Dialogfenster möglich, in dem man eine Zahl an das `T` anhängt.

Das Kommando `Program Step`, (das bei der Einführung in DEBUG 3.0 »Proceed« genannt wurde) ist dem Trace-Kommando ähnlich, überwacht aber Unterprogrammaufrufe und Softwareinterrupts nicht. Dies ist not-

wendig beim Überspringen von BIOS-Aufrufen oder von ausgetesteten Unterprogrammen. Sie können dieses Kommando durch die Eingabe von P im Dialogfenster, dem Drücken der Taste [F10] oder dem Anklicken von F8=Trace mit dem rechten Mausknopf ausführen.

Bedenken Sie, daß Program Step hinter den Aufruf einen Breakpoint setzt. Dieser Breakpoint funktioniert nicht im ROM oder bei Unterprogrammen oder Interrupts, die die Rückkehradresse manipulieren.

Das Go-Kommando führt ein Programm bis zum Erreichen eines Breakpoints aus. Sie können einen Breakpoint zusammen mit der Eingabe des Go-Kommandos eingeben, indem Sie im Dialogfenster G mit dem Breakpoint (als Adresse, Zeilennummer oder symbolisches Label) eingeben. Sie können aber auch den Cursor im Quellfenster an die Stelle positionieren und [F7] drücken. Wünschen Sie Go ohne Breakpoint, drücken Sie [F5]. Mit der Maus können Sie das Programm bis zu einer bestimmten Zeile ausführen, wenn Sie in der Zeile den rechten Mausknopf drücken. Für ein Go ohne Breakpoint klicken Sie F5=Go in der Menüzeile mit der Maus an.

Den Benutzern von SYMDEB ist das explizite Setzen von Breakpoints vertraut. Der Vorteil dieser Methode, gegenüber der Angabe des Breakpoints im Go-Kommando ist, daß Sie mehrere Breakpoints setzen können. Diese bleiben außerdem aktiv, bis sie deaktiviert oder gelöscht werden.

Das Breakpoint-Kommando in CodeView arbeitet wie das in SYMDEB. Zwanzig Breakpoints stehen zur Verfügung. Das Setzen mit der Maus ist einfach, es braucht nur die entsprechende Zeile mit dem linken Mausknopf angeklickt werden. Sie können auch mit der Tastatur den Cursor in die Zeile bewegen und [F9] drücken.

Jenseits von SYMDEB

Lassen Sie uns nun einige Bereiche betrachten, bei denen CodeView die Funktionalität von SYMDEB übertrifft. Es gibt *Watches*, *Watchpoints* und *Tracepoints*. Sie erscheinen alle im selben Menü, sie werden im selben Fenster angezeigt und sie haben leicht zu verwechselnde Namen. Sie sind aber nicht gleich.

Ein *Watch* ist ein Ausdruck, der globale oder gerade verfügbare lokale Variablen enthält. Der Ausdruck wird zusammen mit dem Ergebnis im Watchfenster angezeigt. Wenn Sie zum Beispiel eine Funktion haben, die direkt in den Bildschirmspeicher schreibt und hierzu die Variablen row und col verwendet, könnten Sie Interesse an der aktuellen Bildschirmadresse haben. Ihr Watch schaut dann ungefähr so aus:

```
2 * (row + 80 *col)
```

Der Ausdrucksauswerter kann sogar Aufrufe von Funktionen Ihres Programms handhaben. So können Sie einzelne Funktionen testen, indem Sie diese mit Testwerten aufrufen.

Ein *Watchpoint* ist ein Bool'scher Ausdruck, wiederum in der Syntax der Programmiersprache mit globalen und gerade lokal verfügbaren Variablen. Wird der Watchpoint TRUE (also ungleich 0), stoppt CodeView und übergibt die Kontrolle wieder an Sie.

Sowohl für Watches als auch für Watchpoints muß CodeView immer wieder die Ausdrücke auswerten, während Sie durch das Programm tracen. Für Watches werden nur die neuen Werte angezeigt. Bei Watchpoints gegebenenfalls das Programm abgebrochen.

Ein *Tracepoint* ist ähnlich einem Watchpoint, er kann auch zum Halten des Programms führen. Im Gegensatz zum Watchpoint, der von der Bewertung eines Ausdrucks abhängt, stoppt ein Tracepoint das Programm, wenn ein Teil des Speichers sich ändert. Diese Aufgabe kann CodeView sehr viel leichter und schneller erledigen, aber es ist nicht so flexibel.

Stellen Sie sich eine for-Schleife vor, die die Variable i folgendermaßen erhöht:

```
for(i = 0;i < 100; i++)
```

Wenn Sie bei jeder Änderung von i abbrechen wollen, können Sie einen Tracepoint setzen, der nur der Ausdruck i ist. Ist aber i eine Registervariable, funktioniert der Tracepoint nicht, da i keinem Speicherbereich entspricht.

Wollen Sie bei jedem zehnten Schleifendurchlauf unterbrechen, können Sie keinen Tracepoint setzen, der folgendermaßen aussieht

```
i % 10
```

da i % 10 nicht einem Speicherplatz entspricht. Hier hilft ein Watchpoint der Form:

```
i % 10 == 0
```

Das Beobachten von Tracepoints

Beim Debuggen eines Programms ist es ratsam, daß Sie Ihr Problem lokalisieren, bevor Sie Watch- oder Tracepoints setzen. Watchpoints benötigen länger zum Auswerten als Tracepoints, müssen aber keinem Speicherbereich zugeordnet sein. Es gibt eine einfache Regel. Müssen Sie einen Watch- oder Tracepoint setzen, so sollten Sie es zuerst mit einem Tracepoint versuchen. Funktioniert das nicht, dann nehmen Sie einen Watchpoint.

Watch- und Tracepoints benötigen bei jedem Programmschritt Zeit zur Auswertung. Für Echtzeitanwendungen benötigen Sie daher einen Hardwaredebugger.

Null-Zeiger

Die Möglichkeit, den Inhalt einer Variablen oder einen Speicherbereich ständig zu überwachen, ist eine sehr große Hilfe beim Debuggen. Möglicherweise ist Ihnen die Laufzeitfehlermeldung »error 2001: Null pointer assignment« vertraut. Am Beginn des Datensegments Ihres Programms ist ein kleiner Bereich, der Nullen und die Microsoft Copyright-Meldung enthält. Der Programmteil zum Beenden eines Programms berechnet eine Quersumme. Wurde der Speicherbereich verändert erscheint die Fehlermeldung »Null pointer-assignment«.

In C ist ein Null-Zeiger ein ungültiger Zeiger. Wurde der Beginn des Datensegments verändert, kann dies sehr leicht die Folge der Verwendung eines nicht initialisierten Zeigers sein. Die Meldung sagt nicht aus, daß Sie einem Zeiger 0 zugewiesen haben, sondern daß Sie den Zeiger benutzt haben, um irgendetwas in den Speicher zu schreiben.

Sie können den Programmteil mit dem Fehler finden, indem Sie einen Tracepoint auf den Speicherbereich setzen. Im Dialogwindow können Sie

```
TPB 0 L 8
```

eingeben. Das bedeutet: Setzen eines Tracepoints vom Typ Byte ab Adresse 0 mit einer Länge von 8 Byte. CodeView zeigt die Werte in einem Hexdumpähnlichen Anzeigeformat an. Oder Sie können dasselbe erreichen, wenn Sie im Watchmenü die Option `Tracepoint` auswählen und den Ausdruck

```
*((double *) 0)
```

eingeben. Der Inhalt dieses Zeigers auf die Double-Variable an der Adresse 0 sind die 8 Byte beginnend an Adresse 0. Wenn sich der Speicherbereich ändert, bricht CodeView die Programmausführung ab und zeigt Ihnen den Programmteil, wo es sich ereignete.

Wenn Sie eine Zeichenkette haben, die überschrieben wird, oder ein Array, das sonderbare Werte enthält oder sogar Programmteile, die überschrieben werden, setzen Sie einen Tracepoint auf die Variable oder die Adresse, und CodeView unterbricht beim Auftreten des Überschreibens.

Die Panik-Taste

CodeView unterstützt auf ATs auch den dynamischen Abbruch mit der Taste [SysReq]. Mit [SysReq] können Sie ein Programm wieder unter Kontrolle bekommen, wenn es die Breakpoints, die Sie gesetzt haben, nicht erreicht. Die Option `Calls` im Menü oder das Kommando `K` im Dialogfenster zeigt einen Stacküberblick. Hieran erkennen Sie, wo Sie sind, und auf welchem Weg Sie an diese Programmstelle gelangt sind. Es erfolgt die Anzeige der Funktionsnamen, der übergebenen Parameter, und falls die Eingabe im Dialogfenster erfolgte, die Angabe der Zeilennummer der Stelle, wo der Aufruf erfolgte.

Trotzdem aufpassen

Bei der Verwendung von CodeView sollten Sie einige Vorsicht walten lassen. Wenn Sie längere Zeit DEBUG oder SYMDEB benutzt haben, sollten Sie daran denken, daß CodeView für die Eingabe das Dezimalsystem verwendet. Bei der Eingabe von `D 197F` erhalten Sie eine Fehlermeldung. Sie sollten `D 6527` oder `D 0x197f` eingeben. Sie können aber auch das Zahlensystem auf Hexadezimal ändern, wenn Sie das lieber wollen. (Dumps, Assemblerlistings und Registerinhalte werden immer hexadezimal angezeigt.)

Bedenken Sie jedoch, daß Watches, Watchpoints und Tracepoints dasselbe Zahlensystem benutzen. Setzen Sie die Zahl auf 16, so werden die Zahlen in den Ausdrücken ebenso als hexadezimal angenommen. Schauen wir uns nochmal das Watch von vorher an.

```
2 * (row + 80 * col)
```

Nachdem das Zahlensystem zur Basis 16 eingestellt ist, wird auch die 80 als Hexadezimalzahl (Dezimal 128) behandelt. Wenn Sie das Zahlensystem verändern, schreiben Sie die Ausdrücke mit expliziten hexadezimalen oder dezimalen Zusätzen.

```
2 * (row + 0x50 * col)
```

oder

```
2 * (row + 0n80 * col)
```

SYMDEB ist in einigen Fällen leichter zu handhaben. Wollen Sie sich disassemblierten ROM-Code ansehen, disassemblierte Programmteile ausdrucken oder kurze Assemblerprogramme mit Dateinamenserweiterung .COM erzeugen, sollten Sie SYMDEB verwenden. Für die ernsthafte Fehlersuche aber sollten Sie CodeView einsetzen.

Lassen Sie uns einen Debugvorgang mit CodeView durchführen, um seine Fähigkeiten zu demonstrieren.

Das Debuggen von WHERE

WHERE ist ein in C programmiertes Dienstprogramm, das mit dem Microsoft C-Compiler (Schalter `/Zi`) übersetzt wurde (*Listing 1*). Es benötigt als Argumente einen oder mehrere Programmnamen, und versucht diese Programme genauso wie MS-DOS zu finden. Daher ist die Utility nützlich, um herauszufinden, ob »TEST« »E:\TEST.BAT« oder »C:\BIN\TEST.EXE« bedeutet.

```
 File  View  Search  Run  Watch  Options  Language  Calls  Help | F8=Trace F5=Go
                                   where.c
1:      /* WHERE zeigt wo DOS Ihr ausführbares Programm findet */
2:
3:      #include        <stdio.h>
4:      #include        <time.h>
5:      #include        <sys\types.h>
6:      #include        <sys\stat.h>
7:
8:      #define NUMEXTS 3
9:      #define MAXFILELEN      12
10:     #define MAXPATHLEN      65
11:
12:     struct  stat    file_stat;
13:     char    *getenv(char *);
14:     char    *strchr();
15:
16:     main (argc, argv)
17:     int     argc;
18:     char    **argv;

Microsoft (R) CodeView (R)  Version 2.0
(C) Copyright Microsoft Corp. 1986, 1987.  All rights reserved.
>
```

Bild 2: *Die Eingabe von »T« oder das Anklicken der Menüoption* `F8=Trace` *mit der Maus erzeugt eine blaue Linie im Quelltextfenster in der Zeile 17.*

```
 File  View  Search  Run  Watch  Options  Language  Calls  Help | F8=Trace F5=Go
                                   where.c
_main:
17:     int     argc;
6B23:0010 55            PUSH     BP
6B23:0011 8BEC          MOV      BP,SP
6B23:0013 B85600        MOV      AX,0056
6B23:0016 E87306        CALL     __chkstk (068C)
6B23:0019 57            PUSH     DI
6B23:001A 56            PUSH     SI
23:             char    *ptr = pathbuf;
6B23:001B 8D46BA        LEA      AX,Word Ptr [pathbuf]
6B23:001E 8946FC        MOV      Word Ptr [ptr],AX
27:             if (argc < 2) {
6B23:0021 837E0402      CMP      Word Ptr [argc],+02
6B23:0025 7C03          JL       _main+1a (002A)
6B23:0027 E91400        JMP      _main+2e (003E)
28:                     printf("Bitte WHERE Dateiname[.EXT] eingeben \n");
6B23:002A B83600        MOV      AX,0036
6B23:002D 50            PUSH     AX

Microsoft (R) CodeView (R)  Version 2.0
(C) Copyright Microsoft Corp. 1986, 1987.  All rights reserved.
>u main
>
```

Bild 3: *Das Kommando* `u main` *veranlaßt CodeView sowohl Quellcode- als auch Assembleranweisungen im Quelltextfenster anzuzeigen.*

In der Hoffnung, daß ein wirklicher Programmierer so etwas fehlerfrei erledigt, starten wir WHERE zum ersten Mal und beten, daß es unsere Festplatte nicht zerstört.

```
A>WHERE WHERE.EXE
WHERE.EXE          6224        10-30-86 11.23a
```

Überprüfen wir die Angaben mit dem DIR-Befehl:

```
A>DIR WHERE.EXE
WHERE.EXE          6224        10-30-86 11:23p
```

Anscheinend ist Zeitangabe (a/p) nicht richtig. Ein kurzer Blick auf die Routine `printstats` zeigt uns den Tippfehler. Es sollte dort >= statt <= stehen. Da wir richtige Programmierer sind, merken wir uns den Fehler, und vergessen all die Male, die wir vergessen haben, einen Fehler später zu berichtigen.

WHERE ist noch nicht ausgetestet. Es soll wie DIR auch mehrdeutige Namen finden und soll die Suche in derselben Weise wie MS-DOS durchführen. Die Erweiterungen .COM, .EXE und .BAT werden automatisch angehängt. Ein einfacher Test:

```
A>WHERE WHERE
A>
```

WHERE findet sich selbst nicht. Wie steht es mit einem mehrdeutigen Dateinamen irgendwo im Pfad (nicht im aktuellen Verzeichnis)?

```
A>WHERE COMMAND
A>
A>WHERE COMMAND.COM
A>
```

Anscheinend haben wir zwei Bugs. Wir können nicht beide gleichzeitig finden, also Debuggen wir.

Bild 2 zeigt die anfängliche Bildschirmanzeige von CodeView. Das Registerfenster ist beim Debuggen von C-Programmen geschlossen, und das Watchfenster ist leer. Das Drücken der Taste »T« oder das Anklicken der Menüoption `F8=Trace` mit der Maus läßt eine blaue Linie im Quellfenster in der Zeile 17 erscheinen, wo die aktuelle Programmzeile steht. Das nächste Tracen bringt uns zur Zeile 23.

Sie wundern sich möglicherweise, daß die erste Anweisung in der Zeile nach `main` sein soll, die nächste aber 5 Zeilen später. Der C-Compiler schreibt nur Zeilennummern von Zeilen in die Objektdatei, die ausführbare Befehle enthalten. In den Zeilen 18 bis 22 stehen aber nur Deklarationen, die keinen Code erzeugen, Zeile 23 hingegen enthält eine Zuweisung. Wir können uns mit der Eingabe `u main` den Zusammenhang von Quellcode und erzeugtem Code anschauen. Durch diesen Befehl zeigt CodeView sowohl Quellcode als auch die Assembleranweisungen im Quelltextfenster an. Mit der Taste [F3] gelangen Sie wieder in den Nur-Quellcode-Modus (*Bild 3*).

Einzelschrittablauf

Da es sich um ein recht kurzes Programm handelt, können wir im Einzelschrittmodus durch das Programm gehen, und uns ansehen, was sich ereignet. Die uns interessierenden Variablen können wir im Watchfenster anzeigen lassen, und müssen dadurch nicht jedes Mal den Anzeigebefehl eingeben. Die erste interessante Zeile ist die Zeile 32. Wir können gleich mit `g .32` oder durch das Anklicken der rechten Maustaste dorthin springen. Zur Ergebnisanzeige der Funktion `getenv` geben wir `?envptr` ein. CodeView zeigt das Ergebnis `27998:4641`. CodeView bewertet `envptr` als C-Ausdruck und gibt als Ergebnis den Zeigerwert (die Adresse) zurück.

```
File  View  Search  Run  Watch  Options  Language  Calls  Help | F8=Trace F5=Go
                                where.c
0) envptr,s  :  "c:\msc5\bin;d:\bin\dos"

26:
27:             if (argc < 2) {
28:                     printf("Bitte WHERE Dateiname[.EXT] eingeben \n");
29:                     exit(2);
30:             }
31:
32:             if(!(envptr = getenv("PATH")))
33:                     envptr = "";
34:
35:             /* Schleife durch die angegebenen Dateinamen */
36:             while(ptr = *++argv) {
37:                     strcpy(namebuf, ptr);
38:
39:                     /* Zuerst das aktuelle Verzeichnis versuchen */
40:                     if(!(found = search_for_file(namebuf, strchr(namebuf, '
41:                             /* Schleife durch die Unterverzeichnisse in der

>? envptr
27998:4641
>w?envptr,s
>
```

Bild 4: *Der String, auf den* `envptr` *zeigt.*

```
File  View  Search  Run  Watch  Options  Language  Calls  Help | F8=Trace F5=Go
                                where.c
0) envptr,s  :  "c:\msc5\bin;d:\bin\dos"
1) ptr,s  :  "‡ ╢ ╓o║ "
2) namebuf,s  :  "‡ ╢ ╓o║ .COM"

37:                     strcpy(namebuf, ptr);
38:
39:                     /* Zuerst das aktuelle Verzeichnis versuchen */
40:                     if(!(found = search_for_file(namebuf, strchr(namebuf, '
41:                             /* Schleife durch die Unterverzeichnisse in der
42:                             envpath = envptr;
43:                             while(*envpath) {
44:                                     ptr = pathbuf;
45:                                     /* Pfad in den Puffer kopieren */
46:                                     while(*envpath && *envpath != ';')
47:                                             *ptr++ = *envpath++;
48:
49:                                     /* Pfadtrennung überspringen */
50:                                     while(*envpath == ';')

>w?envptr,s
>t 2;w?ptr,s;w?namebuf,s
>p
>
```

Bild 5: *Das Übergehen der Funktion* `search_for_file` *mit dem Kommando* `P`.

Die Ergebnisse der Ausdrücke `&envptr`, `envptr` und `*envptr` sind unterschiedlich, und CodeView bewertet sie richtig. Uns interessiert der mit 0 abgeschlossene String, auf den `envptr` zeigt. Deshalb müssen wir

```
>w?envptr,s
```

eingeben, um den String im Watch-Fenster angezeigt zu bekommen (*Bild 4*). Zeiger und Zeigerausdrücke in C sind schwer zu erlernen. Unerfahrene Anwender können CodeView in dieser Hinsicht verwirrend empfinden, aber nicht verwirrender, als das Programmieren in C. Ganz im Gegenteil, CodeView ist in dieser Beziehung ein ausgezeichneter Lehrmeister.

Die nächsten beiden Zeilen ändern die Variablen `ptr` und `namebuf`. Tracen wir die Zeilen und lassen uns die Werte im Watchfenster anzeigen. Das darf auch in einer Zeile erfolgen, wenn die Kommandos mit Strichpunkten getrennt werden.

```
>t 2;w?ptr,s;w?namebuf,s
```

Die aktuelle Zeile führt den Funktionsaufruf von `search_for_file` aus. Übergehen wir den Funktionsaufruf mit dem Kommando `P`.

```
>p
```

Die Anzeige entspricht jetzt dem *Bild 5*. Sowohl `envptr` als auch `ptr` sind falsch. Die Funktion `search_for_file` sollte diese zu `main` lokalen Variablen nicht ändern können. Wir müssen die Programmausführung stoppen, wenn diese Variablen geändert werden. CodeView ist uns mit den Kommandos `Watchpoint` und `Tracepoint` behilflich.

Wir wissen nicht, wie die Strings, auf die die Zeiger `envptr` und `ptr` zeigen sich selbst verändern, deshalb müssen wir beide Bedingungen untersuchen. Zuerst starten wir das Programm neu bis dorthin, wo der Fehler auftrat.

```
>l;g .40
```

Zum Anhalten der Ausführung bei Änderungen von `envptr` können wir das folgende Watchpointkommando benutzen.

```
>wp?main.envptr!=4636
```

Wir müssen die Funktion angeben, zu der `envptr` gehört, da wir auf die Variable außerhalb des regulären Gültigkeitsbereichs (`main`) zugreifen. Das Tracepoint-Kommando `tpw envptr` würde auch funktionieren. In beiden Fällen ist CodeView nur wenig behilflich, da der Anwender entweder den Anfangswert von `envptr` oder die Größe des Zeigers wissen muß. Wir können für die zweite Bedingung auch

```
>tpb *ptr
```

eingeben. Dieses Kommando bricht ab, wenn sich das erste Zeichen von `ptr` ändert. Jetzt können wir das Programm durch Anklicken von `F5=Go` ausführen.

Nach ein paar Sekunden - CodeViews Watch- und Tracepoints können den Programmlauf stark verlangsamen, da sie softwaremäßig emuliert werden, obwohl CodeView auch Debughardware unterstützt - erfolgt die Anzeige von Zeile 82. Der gerade ausgeführte Programmteil war ein Aufruf der Bibliotheksfunktion `strcat`. Bevor wir die Bibliotheksfunktion verdächtigen, schauen wir uns das Ergebnis in der Variablen `fullpath` an. *Bild 6* zeigt das Mißgeschick: Die Funktion `search_ for_file` hat mit `strcat` reichlich Dateinamenserweiterungen angehängt.

```
>?fullpath,s
```

```
File View Search Run Watch Options Language Calls Help | F8=Trace F5=Go
                              where.c
0) envptr,s  :  Unknown symbol
1) ptr,s  :  Unknown symbol
2) namebuf,s  :  Unknown symbol
3) main.envptr!-4641  :  1
4) 6D5C:11D5  63 c

78:             else {
79:                     for(next_ext = 0; next_ext < NUMEXTS; ++next_ext) {
80:                             /* .COM, .EXE und .BAT Zusatz versuchen */
81:                             strcat(fullpath, extension[next_ext]);
82:                             if(found = !stat(fullpath, &file_stat))
83:                                     break;
84:                     }
85:             }
86:
87:             return(found);
88:     }
89:

>tpb *ptr
>?fullpath,s
"command.COM.EXE.BAT"
>
```

Bild 6: *Die Funktion* `search_for_file` *hängt die Dateinamenserweiterungen mit* `strcat` *an den Dateinamen an.*

Wir haben die allokierte Länge von `namebuf` überschritten und die Inhalte der anderen lokalen Variablen von `main` überschrieben. Wir wissen, daß `envptr` verändert wurde. Unglücklicherweise ordnet der Microsoft C-Compiler die lokalen Variablen von `main` nicht in der angegebenen Reihenfolge an. In diesem Fall sind sie in der Reihenfolge `envptr`, `ptr`, `found`, `namebuf`, `envpath` und dann `pathbuf` angeordnet, also komplett anders, als in der Deklarationsreihenfolge.

Der Fehler ist schnell beseitigt, indem die vorherige Dateinamenserweiterung mit der Funktion `strchr` am Ende der Dateinamenserweiterungsschleife entfernt wird.

```
*strchr(fullpath,'.') = '\0';
```

Zusammenfassung

CodeView ist nicht neu, etliche seiner Kommandos und Möglichkeiten bieten auch andere Debugger. Neu ist jedoch die Anzahl der Kommandos und das Maß der Integration. Am wichtigsten bleibt aber, daß die Programmierer den Debugvorgang gut verstehen, und daß sie die Programmiersprache und den Debugger bestmöglich für sich einsetzen können.

D. Norris/M. J. O'Leary/Ch. Petzold/ni/jü

Ein Maskengenerator für Microsoft COBOL:

Schnelle COBOL-Bildschirme mit SDA/PC

COBOL ist von Haus aus bereits eine sehr wortreiche Sprache. Bei der Erstellung von interaktiven Bildschirmmasken ist der Eingabeaufwand jedoch überproportional hoch. Hier ist eine Hilfe sehr willkommen. SDA/PC ist eine solche Hilfe, mit der Bildschirmmasken unter COBOL ihren Schrecken verlieren.

Mit SDA/PC stellt sich ein neues Maskengeneratorsystem vor, das speziell auf die Bedürfnisse des Programmierers bei der Arbeit mit Microsoft COBOL ausgelegt ist. Sowohl die Erstellung von komplexen Bildschirmlayouts als auch die Dokumentations- und Testphase werden durch SDA/PC wirkungsvoll unterstützt.

Die wesentliche Funktion von SDA/PC ist die Übersetzung einer auf dem PC-Bildschirm abgebildeten Maske in eine MS-COBOL-kompatible Bildschirmbeschreibung, wie man sie in der `SCREEN SECTION` benötigt. Das bedeutet, daß SDA/PC unabhängig wartbaren Quellcode erzeugt und zur Laufzeit des COBOL-Programms keinerlei zusätzliche Unterstützungen geladen werden müssen.

Die Bildschirmmaske wird direkt vom Bildschirm weg analysiert und übersetzt. Das erfordert relativ wenig Einsatz vom Benutzer. Den größten Teil seiner Zeit verbringt der SDA/PC-Benutzer mit dem Entwerfen und Editieren von Masken, nicht mit dem Codieren. Aus diesem Grund wurde dem eigentlichen Editor besonderes Augenmerk gewidmet. Der SDA/PC-Maskeneditor ist in den wesentlichen Funktionen an den IBM Professional Editor bzw. denn Bildschirmeditor von GW-BASIC angelehnt, besitzt jedoch einige für die Maskenerstellung nützliche Erweiterungen wie Kopier-, Wiederhol- und Zeichenfunktionen.

Für eine Maske steht der gesamte Bildschirmbereich von 24 Zeilen mit 80 Spalten zur Verfügung. Die 25. Zeile wird als Statuszeile verwendet und zeigt die Belegung der Funktionstasten, den Status der Umschalttasten, die aktuelle Schriftfarbe, Cursorposition und Seitennummer an (*Bild 1*). Seitennummer bedeutet hier die Nummer der momentan abgebildeten Bildschirmseite, da mit SDA/PC bis zu vier Masken gleichzeitig bearbeitet werden können. Das Wechseln der aktuellen Bildschirmseite ist jederzeit möglich, auf allen Seiten können alle Funktionen aufgerufen werden, die Seiten sind also vollkommen gleichberechtigt.

Die Kopierfunktionen des Editors erlauben es, Maskenteile zu verschieben, zu kopieren, zu löschen oder neu einzufärben. Verschieben und Kopieren ist auch zwischen verschiedenen Bildschirmseiten möglich, indem nach dem Markieren des Quellblocks einfach die Bildschirmseite gewechselt wird. Dies ermöglicht das Zusammenkopieren von alten Maskenteilen zu neuen Masken, da eine Maske unabhängig von der Übersetzungsfunktion jederzeit gespeichert oder wieder geladen werden kann.

```
* Adressverwaltung *

Nummer: 0000
Name:
Straße:
PLZ:        Ort:

1HILFE 2DATEI 3SEITE 4BLOCK 5ATTR 6SCHIRM 7FARBE 8SATTR 9 DOS 0ENDE  13  25 KB:   1
```

Bild 1: *Die Eingabe einer Bildschirmmaske unter SDA/PC.*

An speziellen Funktionen bietet der Editor unter anderem die Möglichkeit, mit einer Tastenkombination das zuletzt eingegebene druckbare Zeichen (oft mühsam mit `Alt` und dreistelligem Nummerncode eingegeben) beliebig oft zu wiederholen. Als nicht druckbar werden hier alle Positionier-, Lösch- und Kopierfunktionen behandelt.

Für das Zeichnen auf dem Bildschirm erlaubt der Editor drei Modi: Zeichnen mit einfachen und doppelten Linien (auch gemischt) und Zeichnen mit beliebigem (druckbaren) Zeichen. In allen Zeichenmodi wird durch Bewegen des Cursors die gewünschte Linie gezogen.

Für die Definition variabler Maskenfelder verwendet SDA/PC Sonderzeichen als Platzhalter für die häufigsten `PICTURE`-Formatzeichen, das sind »9«, »Z«, »S«, »-«, »,«, ».« und »X«. Unterstützt werden somit Felder vom Typ `ALPHANUMERIC`, `NUMERIC` und `NUMERIC-EDITED`. Für numerische Felder kann wahlweise europäische (`DECIMAL POINT IS COMMA`) oder amerikanische Zahlennotation verwendet werden. Zusätzlich können jedem variablen Feld nach Festlegung des Namens die Zusatzattribute `AUTO`, `BELL`, `SECURE`, `KEY` und `ALTERNATE KEY` verliehen werden sowie der E/A-Modus (`FROM`, `TO` oder `USING`) eingestellt werden.

Die Attribute `KEY` und `ALTERNATE KEY` werden nur beachtet, wenn eine weitere SDA/PC-Funktion verwendet wird - die Erstellung eines Programms. SDA/PC übersetzt dann nicht nur die Maske, sondern verwendet diese gleich bei der automatischen Erstellung eines unmittelbar kompilierbaren, einfachen Dateiwartungsprogramms (*Listing 1*). Diese Möglichkeit erweist sich besonders beim Erstellen von Testdateien als sehr zeitsparend.

Auf Wunsch wird zusätzlich noch eine `FD`-Beschreibung des auf der Maske definierten Datensatzes angefertigt, die analog zu übersetzten Maske etwa mit `COPY` in ein Programm eingefügt werden kann. Zu beachten ist dabei allerdings, daß `PICTURE`-Klauseln vom Typ `NUMERIC-EDITED` automatisch auf `NUMERIC` (also nur aus »S«, »V« und »9« bestehend) konvertiert werden. Auch der `FD` im Dateiwartungsprogramm wird dieser Konvertierung unterworfen.

Alles über die „Internas" der Hardware.

Aus aller Welt: Compaq-Anwender-Stories. Einsatzbeispiele und Problemlösungen.

Alles über das Angebot an Peripherie und Zubehör, das „compaqtibel" ist.

Aus der riesigen Software-Bibliothek für Compaq. Informationen und Empfehlungen zu Auswahl und Einsatz.

Die „Schnittstelle" zum Anwender.

Für das Compaq-Magazin gibt es nur ein Thema: Compaq. Aber das ist ein Thema mit vielen Variationen. Und ein Thema mit vielen Seiten. Compaq-Computer sind mit die meist gekauften professionellen Personal Computer, und der Anwender-Kreis in Wirtschaft, Industrie und Wissenschaft wird größer und größer.

Erfahrungen, Problemlösungen, Anwendungen, Software und Know-How sind vorhanden. Ein riesiger Markt für Software, Peripherie und Zubehör ist entstanden. Neue Ideen, Techniken und Möglichkeiten bieten sich an. Für das Compaq-Magazin sind dies die Themen, um die sich alles dreht. Themen, die den Anwender interessieren, die ihm nützen und ihn mit Informationen versorgen für die Arbeit mit seinem Compaq.

Das Compaq-Magazin ist die „Schnittstelle" zum Anwender.

COMPAQ MAGAZIN

November '87

Das Magazin für den Anwender.

Die Neuen: Deskpro 386/20 Portable 386

Utilities: Die Festplatte optimal nutzen

Windows 386

Eine Zeitschrift aus dem IWT-Verlag.

NEU!

Bestellcoupon

Hiermit bestelle/n ich/wir:

- ☐ das Compaq-Magazin im Abonnement (vorerst 4 Ausgaben pro Jahr) zum Abopreis von DM 40,– (4 x 7,50 zzgl. Versandkosten). Ich kann das Abonnement 8 Wochen vor Ende des Bezugszeitraumes kündigen.
- ☐ ein Einzelheft zum Preis von DM 11,– (DM 8,50 + DM 2,50 Versandkosten)
- ☐ gegen Vorauszahlung auf unser Post-Girokonto München, Kto.-Nr. 99069-804, BLZ 700 100 80
- ☐ gegen Nachnahme ☐ gegen Rechnung

Meine/unsere Anschrift:

Name

Firma

Straße

Ort

Telefon

Datum, Unterschrift

IWT Magazin Verlags GmbH
Wendelsteinstr. 3, 8011 Vaterstetten

```
IDENTIFICATION DIVISION.
    PROGRAM-ID. SAMPLE.
    AUTHOR. SDA-PC.
ENVIRONMENT DIVISION.
SOURCE-COMPUTER. IBM-PC-XT-AT.
OBJECT-COMPUTER. IBM-PC-XT-AT.
SPECIAL-NAMES.
    DECIMAL-POINT IS COMMA.
FILE-CONTROL.
SELECT DF ASSIGN TO DISK
    ORGANIZATION IS INDEXED
    ACCESS IS DYNAMIC
    RECORD KEY IS ADDRESS-0
    ALTERNATE RECORD KEY ADDRESS-1
    ALTERNATE RECORD KEY ADDRESS-3
    FILE STATUS IS F-STAT.
DATA DIVISION.
FILE SECTION.
FD DF LABEL RECORD STANDARD VALUE OF FILE-ID 'ADDR.DAT'.
COPY SAMPLE.REC.
WORKING-STORAGE SECTION.
77 F-STAT PIC XX.
77 TMP PIC XX.
77 NEWF PIC 9 VALUE 0.
77 CHAR PIC X.
77 MSG PIC X(40).
SCREEN SECTION.
01 BLANK-SCREEN.
    02 BLANK SCREEN.
01 BLANK-LINE.
    02 LINE 25 COLUMN 1 BLANK LINE.
01 MENU.
    02 LINE 25 COLUMN 1 REVERSE-VIDEO VALUE
       ' Satz gefunden - Wählen Sie <Ä>ndern oder <L>öschen : '.
    02 PIC X TO CHAR AUTO.
    02 REVERSE-VIDEO VALUE '                          '.
01 NEWREC.
    02 LINE 25 COLUMN 1 REVERSE-VIDEO VALUE
       ' Neuer Satz - Kein Datensatz mit diesem KEY gefunden.
-      '                 '.
01 ERR-SCREEN.
    02 LINE 25 COLUMN 1 REVERSE-VIDEO VALUE ' FEHLER: '.
    02 PIC X(40) FROM MSG REVERSE-VIDEO.
    02 REVERSE-VIDEO VALUE ' - Bel. Taste drücken '.
    02 PIC X TO CHAR BELL AUTO.
    02 REVERSE-VIDEO VALUE '      '.
01 KEY-SCREEN.
    02 LINE 11 COLUMN 25 PIC 9999 USING ADDRESS-0 BELL.
01 ADDRESS-SCREEN.
    02 LINE 2 COLUMN 27 VALUE '┌──────────────────────┐'.
    02 LINE 3 COLUMN 27 VALUE '│ * Adressverwaltung * │'.
    02 LINE 4 COLUMN 27 VALUE '└──────────────────────┘'.
    02 LINE 7 COLUMN 23 VALUE '┌──────────────┐'.
    02 LINE 8 COLUMN 20 VALUE '┌──┴──────────────┐  └──────────────
-   '──────────────┐'.
    02 LINE 9 COLUMN 17 VALUE '┌──┴──────────────┐  └──────────────
-   '──────────────┐ │'.
    02 LINE 10 COLUMN 15 VALUE '╔═┴═══════════════╗  └─────────────
-   '──────────────┐ │ │'.
    02 LINE 11 COLUMN 15 VALUE '║ Nummer: '.
    02 PIC 9999 FROM ADDRESS-0 BELL.
    02 COLUMN 30 VALUE '╚══════════════════════════╗ │ │ │'.
    02 LINE 12 COLUMN 15 VALUE '║'.
    02 COLUMN 59 VALUE '║ │ │ │'.
    02 LINE 13 COLUMN 15 VALUE '║ Name:'.
    02 COLUMN 25 PIC X(20) USING ADDRESS-1.
    02 COLUMN 59 VALUE '║ │ │ │'.
    02 LINE 14 COLUMN 15 VALUE '║'.
    02 COLUMN 59 VALUE '║ │ │ │'.
    02 LINE 15 COLUMN 15 VALUE '║ Straße: '.
    02 PIC X(20) USING ADDRESS-2.
    02 COLUMN 59 VALUE '║ │ ├─┘'.
    02 LINE 16 COLUMN 15 VALUE '║'.
    02 COLUMN 59 VALUE '║ ├─┘'.
    02 LINE 17 COLUMN 15 VALUE '║ PLZ: '.
    02 PIC X(8) USING ADDRESS-3.
    02 COLUMN 33 VALUE 'Ort: '.
    02 PIC X(20) USING ADDRESS-4.
    02 COLUMN 59 VALUE '║─┘'.
    02 LINE 18 COLUMN 15 VALUE '╚══════════════════════════════════
-   '════════════╝'.

PROCEDURE DIVISION.
W1.
    DISPLAY BLANK-SCREEN.
    OPEN I-O DF.
    IF F-STAT = '00' GO TO W2 ELSE MOVE F-STAT TO TMP CLOSE DF.
    IF TMP = '30' OPEN OUTPUT DF MOVE 1 TO NEWF GO TO W2 ELSE
    MOVE 'ISAM-Datei läßt sich nicht öffnen' TO MSG GO TO ERR2.
W2.
    PERFORM INIT-F DISPLAY ADDRESS-SCREEN KEY-SCREEN.
    ACCEPT KEY-SCREEN ON ESCAPE GO TO ENDP.
    IF NEWF = 1 GO TO W5.
    READ DF RECORD INVALID GO TO W5.
W3.
    DISPLAY ADDRESS-SCREEN.
W31.
    DISPLAY MENU ACCEPT MENU ON ESCAPE GO TO W2.
.
.
.
```

Listing 1: *Ein Auszug aus einem mit SDA/PC erzeugtem Dateneingabeprogramm. Auf Wunsch kann auch nur der Bildschirmaufbau erzeugt werden (ADDRESS-SCREEN).*

```
* Adressverwaltung *

Nummer: 0002
Name:   IKARUS Software GmbH
Straße: Franz-Josef-Str. 7
PLZ: A-5700      Ort: Zell am See

Satz gefunden - Wählen Sie <Ä>ndern oder <L>öschen :
```

Bild 2: *So läuft das von SDA/PC generierte Programm (Listing 1) für die Beispiel-Bildschirmmaske (Bild 1).*

An Bildschirmadaptern werden sowohl die Monochrom- als auch die Farbkarten unterstützt. Sind beide Arten am PC angeschlossen, kann jederzeit der Bildschirm gewechselt werden. In jedem Fall stellt sich SDA/PC beim Start auf den momentan aktiven Bildschirm ein, da es bezüglich der Behandlung der Bildschirmattribute für jede der beiden Karten einen anderen Übersetzungsmodus gibt. Dieser Modus kann jederzeit von Hand geändert werden, also die gleiche Maske einmal im Farb- und einmal im Monochrom-Modus übersetzt werden. Der vielseitigere ist der Farbmodus, da durch trickreiche Codierung der Attribute sichergestellt wird, daß auch eine im Farbmodus übersetzte Maske die Möglichkeiten des monochromen Bildschirms optimal verwendet.

Ein positiver Nebeneffekt der Verwendung von SDA/PC ist überdies eine gewisse Speicherplatzersparnis. Das gilt sowohl für den Massenspeicher, wo Masken in einem speziellen, komprimierten Format abgelegt werden, als auch für das einzelne COBOL-Programm, da SDA/PC-codierte Masken oft weniger Platz im Codesegment belegen und schneller ausgegeben werden, als manuell codierte.

Im Lieferumfang enthalten ist ein Hardcopy-Programm, das resident geladen wird und das unter anderem die Möglichkeit bietet, den momentanen Bildschirminhalt in einer SDA/PC-Maskendatei zur späteren Bearbeitung abzulegen.

»Arbeitsersparnis durch Geschwindigkeit und Leistung« war die Devise bei der Entwicklung dieses Werkzeugs für den COBOL-Programmierer, und dieses Ziel ist, denke ich, erreicht worden.

Joachim Schneider/jü

Joachim Schneider ist der Autor von SDA/PC. Weitere Informationen erhalten Sie bei:
IKARUS Software GmbH, Franz-Josef-Str. 7, A-5700 Zell am See, Tel.: (Österreich) 06542/2941.

PC-KNOW-HOW

SCHIEB
DAS GROSSE BUCH ZU
MS-DOS
UND PC-DOS
INKLUSIVE VERSION 3.3
DATA BECKER

TISCHER
INTERN
SYSTEMPROGRAMMIERUNG
DATA BECKER

SCHIEB · TISCHER
DAS GROSSE BUCH ZU
OS/2
DATA BECKER

Das Anwenderhandbuch zu MS-DOS.

Das große Buch zu MS-DOS/PC-DOS vermittelt Ihnen das Detailwissen, mit dem Sie das Betriebssystem optimal für Ihre tägliche Arbeit nutzen können. Ob zu Batch-Dateien, zu den verschiedenen Kopierbefehlen oder zur Konfigurationsdatei – mit diesem Buch wissen Sie immer, worauf es ankommt. Beispielsweise, was Sie beim Anschluß eines zusätzlichen 3½"-Laufwerks beachten müssen. Das große Buch zu MS-DOS/PC-DOS gibt Ihnen auf jede Frage die richtige Antwort – zu allen DOS-Versionen. Denn einschließlich der Version 3.3 werden alle Kommandos ausführlich mit Syntax, Erklärungen und Beispielen aufgeführt. Und selbst wenn Sie am Ende alles über das Betriebssystem MS-DOS wissen: Mit diesem Buch haben Sie auch für Ihre weitere Arbeit mit Ihrem PC ein überaus hilfreiches Nachschlagewerk.

Jörg Schieb
Das große Buch zu MS-DOS/PC-DOS
Hardcover, 427 Seiten
DM 49,–

PC-Know-how vom Profi.

Wer die Intern-Bücher von DATA BECKER kennt, weiß, was ihn mit PC Intern erwartet: Information pur. Das gesammelte Wissen zum PC wird hier aufbereitet – sei es über Hardware, BIOS oder DOS. Nicht nur Lehrbuch, sondern auch Nachschlagewerk von bleibendem Wert. Ein kleiner Streifzug durch das über 700 Seiten starke Werk macht es deutlich: Aufbau der Hauptplatine, Registersatz des Prozessors, DMA-Controller, mathematische Coprozessoren, Hard- und Software-Interrupts, Aufruf von Interrupts in Assembler, BASIC, Pascal und C, die Funktionen des DOS, COM- und EXEC-Funktionen, RAM-Speicherverwaltung, DOS-Gerätetreiber, Booten des Systems, Zugriff auf die Festplatte, die Ports des PCs und, und, und. PC Intern – geschrieben von einem erfahrenen Software-Entwickler. Know-how aus erster Hand.

Michael Tischer
PC Intern
Hardcover, 767 Seiten, DM 69,–

OS/2 – das Betriebssystem der Zukunft.

Die gesamte Fachpresse spricht von OS/2 als dem Betriebssystem der Zukunft für die PC-Welt. Das große Buch zu OS/2 zeigt Ihnen den Einstieg. Sie erfahren alles über die Eigenarten, die Benutzeroberfläche, das Installieren, die Befehle, das Erstellen von Batchprogrammen und das Konfigurieren des Systems durch die Datei CONFIG.SYS. Aber auch die Programmierer und weitergehend Interessierten kommen nicht zu kurz: was sind Threads, Prozesse, Semaphore, Tasks, Prioritäten und Zeitscheiben, wie wird unter OS/2 programmiert, was muß beachtet werden und was ist neu? So ist zum Beispiel Multitasking ein Schlagwort, das sicherlich jeden an OS/2 besonders interessiert. Am besten, Sie lesen einfach mal rein, ins große OS/2-Buch!

Jörg Schieb/Michael Tischer
Das große OS/2-Buch
Hardcover, 461 Seiten, DM 49,–

COUPON

HIERMIT BESTELLE ICH

NAME, VORNAME

STRASSE

ORT

Zuzüglich Versandkosten unabhängig von der bestellten Stückzahl
☐ per Nachnahme ☐ Verrechnungsscheck anbei

DATA BECKER
Merowingerstr. 30 · 4000 Düsseldorf · Tel. (0211) 310010

Faszination Programmieren

Satire, Lesestoff, Computergeschichten, künstliche Intelligenz

S. Lammers
Faszination Programmieren
1987, 430 Seiten

»Faszination Programmieren« ist eine Sammlung aufschlußreicher Interviews mit neunzehn namhaften Programmierern unserer Zeit. Dje Interviews bieten einen Einblick in die Karriere jedes einzelnen, beinhalten Programmskizzen und Quellcode-Beispiele, beleuchten die Motivation und die Programmierbedingungen und schildern nebenbei die Geschichte der Mikrocomputerindustrie. Sie erfahren dabei, wie erfolgreiche Programmierer an ihre Aufgabe herangehen, ob Programmieren eine intuitive oder erlernte Fähigkeit ist, welche Entwicklungsmethodik Programmen wie VisiCalc, Microsoft Basic oder Lotus den Erfolg gebracht haben, die Hintergründe der Entstehung von Firmen und Forschungslabors wie Lotus, Apple, Xerox, PARC und Microsoft und vieles mehr.

• Ein packendes Buch – unerläßlich für den Fachmann und fesselnd für den Laien.

Bestell-Nr. 90418
ISBN 3-89090-418-1
DM49,–/sFr 45,10/öS 382,20

Markt&Technik
Faszination Programmieren

Markt&Technik
Pamela McCorduck
DENK-MASCHINEN
Die Geschichte der künstlichen Intelligenz

P. McCorduck
Denkmaschinen
1987, 344 Seiten

»Elektronengehirn«, so lautet eine alte, volkstümliche Bezeichnung für den Computer. Elektronengehirne waren noch raumfüllende Maschinen, und ihre »Intelligenz« hielt sich in engeren Grenzen als die eines heutigen Heimcomputers. Trotzdem haben die Computer vom Anfang ihrer Entwicklung an den Ruf gehabt, intelligent zu sein.
Dieses Buch erzählt die Geschichte dieser »Denkmaschinen«, die Geschichte der »künstlichen Intelligenz«. Die Autorin, Journalistin und Schriftstellerin, hat viele der wichtigsten Persönlichkeiten dieser Geschichte interviewt. Sie erzählt so auch die Geschichte der Menschen, die an dieser Entwicklung Anteil hatten. Das sind vor allem Ada Lovelace und Charles Babbage, Alan Turing, John von Neumann, Minsky, Newell, Simon und viele andere.

• Ein Buch, das sich an den Leser wendet, der mehr über die KI wissen will als Programmstrukturen und Sprachnormen.

Bestell-Nr. 90419
ISBN 3-89090-419-X
DM49,–/sFr 45,10/öS 382,20

K. Kupfernagel
Computer – die leisen Eroberer
1987, 269 Seiten

Was wissen Sie eigentlich über Computer? Kennen Sie seine Möglichkeiten und seine Funktionen? Was ist eigentlich eine Datenbank? Und, und, und. Wenn Sie sich diese und noch mehr Fragen stellen, dann ist das Buch »Computer – die leisen Eroberer« genau die richtige Informationsquelle für Sie. Es gibt Ihnen einen generellen Überblick zum Thema Computer und erläutert seine Fähigkeiten und Funktionsweisen. Ein Computer funktioniert nur nach wenigen Grundprinzipien – sie sind überraschend einfach! Wer diese Prinzipien verstanden hat, kann sich auch sinnvoll mit dem Computer auseinandersetzen. Mit einem Computer umzugehen ist heute wichtiger denn je; denn die Computerrevolution hat wahrscheinlich noch gar nicht richtig begonnen...

Bestell-Nr. 90179, ISBN 3-89090-179-4
DM 14,80/sFr 13,60/öS 115,44

Markt&Technik
Computer - die leisen Eroberer
Wie sie funktionieren, wo ihre Grenzen liegen.

E. Pawlu
Wenn der Computer Geschichten macht
1986, 161 Seiten

Millionen Zeitungsleser kennen Erich Pawlu, der die Entwicklungen in der Computertechnik mit Humor und Scharfblick verfolgt. Sein Buch »Wenn der Computer Geschichten macht« überträgt die Freuden und Probleme des elektronischen Zeitalters in amüsante Geschichten. Mit hintergründiger Satire verfolgt er die Veränderungen und die Möglichkeiten des Komischen, die durch die neuen Computer in unserer Umwelt und in unseren Köpfen entstehen. Einen besonderen Reiz bezieht dieser Band aus den nostalgischen Bildern, die mit »modernisierten« Bildtiteln einen zusätzlichen humorvollen Akzent setzen.

• Ein Buch für alle Computerfreunde und Computergegner die dem Computerzeitalter eine hintergründig-heitere Seite abgewinnen wollen.

Bestell-Nr. 90378, ISBN 3-89090-378-9
DM24,80/sFr 23,–/193,40)

Markt&Technik-Produkte erhalten Sie bei Ihrem Buchhändler, in Computer-Fachgeschäften oder in den Fachabteilungen der Warenhäuser.

Markt&Technik
Zeitschriften · Bücher
Software · Schulung

Markt&Technik Verlag AG, Buchverlag, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0.
SCHWEIZ: Markt&Technik Vertriebs AG, Kollerstrasse 3, CH-6300 Zug, Telefon (042) 415656,
ÖSTERREICH: Markt&Technik Verlag Gesellschaft m.b.H., Große Neugasse 28, A-1040 Wien, Telefon (0222) 5871393-0, Rudolf Lechner&Sohn, Heizwerkstraße 10, A-1232 Wien, Telefon (0222) 677526.

COMPUTER-LITERATUR UND SOFTWARE FRÜHJAHR/SOMMER '88

Fragen Sie bei Ihrem Buchhändler nach unserem kostenlosen Gesamtverzeichnis mit über 500 aktuellen Computerbüchern und Software. Oder fordern Sie es direkt beim Verlag an!

804186